Σχολή Θετικών Επιστημών και Τεχνολογίας

Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

Εφαρμογή Σύγχρονων Τεχνικών Υπολογιστικής Νοημοσύνης Particle Swarm Optimization (PSO) για την Αποδοτική Επίλυση του Προβλήματος Sports Timetabling

Αθανάσιος Ν. Κρανιδιώτης

Επιβλέπων καθηγητής: Γρηγόριος Μπεληγιάννης

Πάτρα, Σεπτέμβριος 2024

Εφαρμογή Σύγχρονων Τεχνικών Υπολογιστικής Νοημοσύνης Particle Swarm Optimization (PSO) για την Αποδοτική Επίλυση του Προβλήματος Sports Timetabling

Αθανάσιος Ν. Κρανιδιώτης

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Γρηγόριος Μπεληγιάννης

Καθηγητής Πανεπιστημίου Πατρών

Τμήμα Επιστήμης και Τεχνολογίας Τροφίμων

Συν-Επιβλέπων Καθηγητής:

Βασίλειος Καψάλης

Καθηγητής Πανεπιστημίου Πελοποννήσου

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πάτρα, Σεπτέμβριος 2024

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου, κύριο Γρηγόριο Μπεληγιάννη για την ανάθεση του θέματος και την εμπιστοσύνη που έδειξε στο πρόσωπό μου καθώς και για την υποστήριξη που μου παρείχε καθ' όλη την διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας.

Περίληψη

Ο χρονοπρογραμματισμός αθλητικών αγώνων είναι ένα δύσκολο πρόβλημα που συγκαταλέγεται συνήθως στην οικογένεια των NP-hard προβλημάτων. Το πρόβλημα αφορά στην δημιουργία ενός ωρολογίου προγράμματος αθλητικών αγώνων που να ικανοποιεί πλήρως ένα σύνολο από δομικούς και ισχυρούς περιορισμούς, ενώ παράλληλα πρέπει να ελαχιστοποιεί το κόστος παραβίασης μιας σειράς από ασθενείς περιορισμούς. Οι δομικοί και οι ισχυροί περιορισμοί ενός προβλήματος χρονοπρογραμματισμού είναι απαραίτητο να ικανοποιούνται πάντα, ενώ οι ασθενείς περιορισμοί είναι επιθυμητό, αλλά όχι απαραίτητο.

Ο χρονοπρογραμματισμός αθλητικών αγώνων είναι ένα υπολογιστικά πολύ απαιτητικό πρόβλημα. Δεν υπάρχει μέχρι σήμερα κάποιος γνωστός και αποδοτικός τρόπος εύρεσης μίας βέλτιστης λύσης του προβλήματος. Εξαιτίας της ιδιαίτερης φύσης και πολυπλοκότητάς του, συνήθως επιστρατεύονται ευρετικές και μεταευρετικές όπως οι Γενετικοί Αλγόριθμοι, ο αλγόριθμος Simulated Annealing (SA), ο αλγόριθμος Tabu Search κ.ά. για την εύρεση μιας καλής και αποδοτικής λύσης σε λογικά πλαίσια χρόνου.

Στην παρούσα εργασία επιχειρείται η αποδοτική επίλυση του προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων με την βοήθεια του υβριδικού αλγορίθμου CP-PSO, ο οποίος βασίζεται στον αλγόριθμο υπολογιστικής νοημοσύνης, Particle Swarm Optimization (PSO) και σε μηχανισμούς Constraint Programming (CP). Ο υβριδικός αλγόριθμος που προτείνουμε επιμερίζει την διαδικασία επίλυσης του προβλήματος σε δύο φάσεις: την φάση ικανοποιησιμότητας (*Satisfiability*) και την φάση βελτιστοποίησης (*Optimization*). Στην πρώτη φάση, υιοθετείται ένα τροποποιημένο μοντέλο που βασίζεται στον αλγόριθμο ανοικτού κώδικα της Google, CP-SAT με την βοήθεια του οποίου παράγεται ένα σύνολο από αρχικά (κανονικά ή εφικτά) σωματίδια. Τα σωματίδια αυτά αποτελούν τα δεδομένα της δεύτερης φάσης, όπου ένα βελτιωμένο μοντέλο PSO εξερευνά τον χώρο αναζήτησης εφαρμόζοντας στρατηγική SA, και τα αποτελέσματα της εξερεύνησης αυτής βελτιώνονται με έναν τροποποιημένο μηχανισμό Hill Climbing.

Ο υβριδικός αλγόριθμος CP-PSO εφαρμόστηκε στα δημοσιευμένα προβλήματα του διαγωνισμού ITC2021 που αποτελεί την καλύτερη βάση δεδομένων προβλημάτων χρονοπρογραμματισμού αθλητικών αγώνων. Παρά την μεγάλη υπολογιστική δυσκολία των

προβλημάτων αυτών και τους περιορισμένους υπολογιστικούς πόρους που είχαμε στην διάθεσή μας, ο αλγόριθμος CP-PSO κατάφερε να επιτύχει καλά αποτελέσματα.

Λέξεις – Κλειδιά

Αλγόριθμοι Υπολογιστικής Νοημοσύνης, Χρονοπρογραμματισμός, Βελτιστοποίηση

Effective Solution of the Sports Timetabling Problem Using Computational Intelligence Particle Swarm Optimization (PSO) Algorithms

Athanasios N. Kranidiotis

Abstract

The Sports Timetabling Problem (STP) is a challenging problem that is often classified as a NP-hard problem. The STP problem involves creating a timetable for organizing matches in a sports league that fully satisfies a set of structural and hard constraints, while minimizing the cost of violation of a series of soft constraints. The structural and hard constraints of a STP problem must always be satisfied, whereas the soft constraints are desirable but not necessary.

The STP problem is a computationally intensive problem. There is no known efficient way for finding an optimal solution to this problem so far. Due to its unique nature and complexity, heuristic and metaheuristic methods such as Genetic Algorithms, Simulated Annealing (SA), Tabu Search, etc., are usually employed to find a good and efficient solution within a reasonable amount of time.

In this work, we approach efficiently the STP problem with the help of the hybrid CP-PSO algorithm, using Computational Intelligence Particle Swarm Optimization (PSO) and Constraint Programming (CP) heuristic. The proposed hybrid algorithm divides the problem-solving process into two phases: the satisfiability phase and the optimization phase. In the first phase, an advanced model based on Google's open-source OR-Tools, CP-SAT, is used for obtaining a set of initial (canonical or feasible) particles. These particles will be the initial data of the second phase, where an advanced PSO model

explores the search space implementing a SA scheme. The output of the exploration and exploitation derived is getting improved with the means of a modified Hill Climbing mechanism.

The CP-PSO hybrid algorithm is tested to the published instances of the ITC2021 competition, which represents the best, well-known database of sports timetabling instances. Despite the significant computational difficulty of these problems and the limited computational resources we had available, the CP-PSO algorithm managed to achieve good results.

Keywords

Particle Swarm Optimization, Sports Timetabling Problem, Scheduling, Optimization

Περιεχόμενα

Κατάλογος Εικόνων / Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες & Ακρωνύμια

PSO	Particle Swarm Optimization
BPSO	Binary Particle Swarm Optimization
DPSO	Discrete Particle Swarm Optimization
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
SA	Simulated Annealing
STP	Sports Timetabling Problem
ITC2021	International Timetabling Competition on Sports Timetabling

1 Εισαγωγή

Στην παρούσα διπλωματική εργασία μελετάται το πρόβλημα του χρονοπρογραμματισμού αθλητικών αγώνων (STP) εφαρμόζοντας τεχνικές υπολογιστικής νοημοσύνης Particle Swarm Optimization (PSO) σε συνδυασμό με άλλες ευρετικές και μετευρετικές μεθόδους (Constraint Programming, Simulated Annealing, Hill Climbing) για την αποδοτική επίλυση του προβλήματος. Τα πειραματικά δεδομένα που χρησιμοποιήθηκαν αντλήθηκαν από την δημόσια βάση δεδομένων του διεθνούς διαγωνισμού ITC2021.

Ο προτεινόμενος υβριδικός αλγόριθμος CP-PSO δοκιμάστηκε σε όλα τα προβλήματα του διαγωνισμού και έγινε αντιπαραβολή των πειραματικών αποτελεσμάτων του αλγορίθμου CP-PSO με τα αποτελέσματα δημοσιευμένων λύσεων του διαγωνισμού και άλλων ερευνητών.

Μάλιστα, σε ορισμένα προβλήματα του διαγωνισμού ITC2021, ο αλγόριθμος CP-PSO κατάφερε να βρει την καλύτερη μέχρι στιγμής και δημοσιευμένη στην ιστοσελίδα του διαγωνισμού λύση, με ίδια τιμή αντικειμενικής συνάρτησης αλλά με διαφορετικό ωρολόγιο πρόγραμμα. Αυτό σημαίνει πως:

- είτε το πρόβλημα έχει παραπάνω από ένα ολικά βέλτιστα και η λύση που βρήκαμε είναι ένα από αυτά,
- είτε το ολικό βέλτιστο του προβλήματος δεν έχει βρεθεί ακόμα και η λύση που ανακαλύψαμε είναι ένα από πολλά τοπικά βέλτιστα.

Κάθε μία από τις παραπάνω περιπτώσεις «αποδεικνύει» πειραματικά ότι το πρόβλημα είναι ιδιαίτερα δύσκολο και υπάρχει μεγάλη δυσκολία στην εύρεση μίας καλής λύσης.

1.1 Διάρθρωση

Η εργασία οργανώνεται σε κεφάλαια ως εξής:

Στο κεφάλαιο 2 γίνεται μια παρουσίαση του αλγορίθμου PSO, της κανονικής μορφής του και μετέπειτα παραλλαγών που εστιάζουν σε προβλήματα βελτιστοποίησης στον διακριτό χώρο.

Στο κεφάλαιο 3 εξετάζονται ο αλγόριθμος τοπικής αναζήτησης Hill Climbing και ο αλγόριθμος βελτιστοποίησης Simulated Annealing, η συνδρομή των οποίων είναι καθοριστική στην ανάπτυξη του αλγορίθμου CP-PSO. Συνάμα, γίνεται μια επιγραμματική αναφορά στα προβλήματα CP, τις τεχνικές επίλυσής τους, στο πρόβλημα ικανοποιησιμότητας (SAT) και στον αλγόριθμο επίλυσης CP-SAT που ανήκει στην οικογένεια εργαλείων βελτιστοποίησης (OR-Tools) ανοικτού κώδικα της Google.

Στο κεφάλαιο 4 γίνεται μια αναλυτική παρουσίαση του προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων, όπου κατηγοριοποιούνται οι διάφορες παράμετροι και οι γνωστές κατηγορίες περιορισμών που θα συναντήσουμε αργότερα στα προβλήματα του διαγωνισμού ITC2021.

Στο κεφάλαιο 5 παρουσιάζεται και αναλύεται το μαθηματικό μοντέλο που θα χρησιμοποιηθεί για την ανάπτυξη του υβριδικού αλγορίθμου CP-PSO.

Στο κεφάλαιο 6 παρουσιάζεται ο προτεινόμενος, υβριδικός αλγόριθμος CP-PSO και αναλύεται εκτενώς η διάρθρωσή του και ο τρόπος λειτουργίας του.

Στο κεφάλαιο 7 παρουσιάζεται αναλυτικά η υλοποίηση του αλγορίθμου CP-PSO σε γλώσσα Python και δίνεται ο κώδικας.

Στο κεφάλαιο 8 εφαρμόζεται ο αλγόριθμος CP-PSO στα προβλήματα του διαγωνισμού ITC2021, αναλύονται τα αποτελέσματα και γίνεται μια συνολική αξιολόγηση του αλγορίθμου. Επίσης, αποτυπώνονται τα συμπεράσματα της μελέτης, οι περιορισμοί της τρέχουσας εργασίας και προτάσεις για μελλοντική έρευνα.

□

2 Particle Swarm Optimization

2.1 Εισαγωγή

Η Τεχνητή Νοημοσύνη (*Artificial Intelligence*) είναι ένας κλάδος της πληροφορικής που ασχολείται με την εξομοίωση της ανθρώπινης νοημοσύνης σε υπολογιστικά συστήματα, τα οποία εκδηλώνουν ιδιότητες όπως αυτόν της μάθησης και της επίλυσης προβλημάτων. Κλάδος της Τεχνητής Νοημοσύνης είναι η Υπολογιστική Νοημοσύνη (*Computational Intelligence*) με πεδία εφαρμογής:

- τα Νευρωνικά Δίκτυα (που δημιουργούν το μοντέλο των δεδομένων και παράγουν προβλέψεις)
- την Fuzzy Logic (που παρουσιάζει την αβεβαιότητα του μοντέλου των δεδομένων)
- την Εξελικτική Υπολογιστική (που χρησιμοποιεί εξελικτικούς αλγορίθμους για βελτιστοποίηση προβλημάτων).

Στον κλάδο της Εξελικτικής Υπολογιστικής, οι πιο γνωστοί αλγόριθμοι είναι οι:

- Particle Swarm Optimization (PSO)
- Γενετικοί αλγόριθμοι
- Αλγόριθμος Hill Climbing
- Αλγόριθμος Simulated Annealing
- Αλγόριθμος Tabu Search
- Αλγόριθμος Great Deluge
- Αλγόριθμος Ant Colony Optimization
- Αλγόριθμος Cuckoo Search
- Αλγόριθμος Variable Neighborhood Search

και άλλοι.

Από τους παραπάνω αλγορίθμους, η παρούσα εργασία θα εστιάσει στον αλγόριθμο PSO και στην εφαρμογή του συνδυαστικά με μηχανισμούς που χρησιμοποιούν στρατηγικές βελτίωσης Hill Climbing, Simulated Annealing και Constraint Programming για την βελτιστοποίηση ενός διακριτού προβλήματος, όπως είναι το πρόβλημα χρονοπρογραμματισμού αθλητικών αγώνων.

2.2 Ορισμός

Ο αλγόριθμος PSO είναι μία υπολογιστική μέθοδος βελτιστοποίησης ενός προβλήματος που λειτουργώντας επαναληπτικά βελτιώνει μία υποψήφια λύση χρησιμοποιώντας ένα μέτρο ποιότητας.

Ο αλγόριθμος PSO αναπτύχθηκε για πρώτη φορά από τους Eberhart & Kennedy (1995). Ο James Kennedy ήταν κοινωνικός ψυχολόγος, ενώ ο Russell Eberhart ήταν ηλεκτρολόγος μηχανολόγος. Εμπνεύστηκαν τον νέο αλγόριθμο από την κοινωνική συμπεριφορά ομάδων οργανισμών, όπως ενός σμήνους πτηνών ή ενός κοπαδιού ψαριών. Οι ερευνητές παρατήρησαν ότι οι οργανισμοί αυτοί είχαν την ικανότητα να συγχρονίζουν τις κινήσεις τους και να προσαρμόζονται στο περιβάλλον τους συλλογικά και χωρίς κάποιο κεντρικό έλεγχο.

Βασική έννοια στον αλγόριθμο PSO είναι το **σμήνος** (*swarm*) που αποτελείται από εν δυνάμει λύσεις που ονομάζονται **σωματίδια** (*particles*). Ο αλγόριθμος PSO αρχικοποιείται με ένα σύνολο από τυχαία σωματίδια και στην συνέχεια αναζητά βέλτιστες λύσεις εξελισσόμενος από **γενιά** (*generation*) σε γενιά στον **χώρο αναζήτησης** (*search space*). Σε κάθε γενιά, κάθε σωματίδιο του σμήνους ενημερώνει την θέση του λαμβάνοντας υπ' όψιν την καλύτερη θέση που έχει αποκτήσει μέχρι στιγμής (την οποία αποθηκεύει στην μνήμη του) και την καλύτερη θέση που έχει βρεθεί από ολόκληρο το σμήνος. Με αυτό τον τρόπο μεταβάλλει την ταχύτητά του και επηρεάζει την κίνηση ολόκληρου του σμήνους που σε βάθος χρόνου αναμένεται να συγκλίνει προς μία βέλτιστη λύση στον χώρο αναζήτησης. Κάθε σωματίδιο έχει μία τιμή κόστους που καθορίζεται από μία **αντικειμενική συνάρτηση ή συνάρτηση στόχου** (*objective function*), η οποία πρέπει να βελτιστοποιηθεί. Η αντικειμενική συνάρτηση είναι το κριτήριο που καθορίζει την ποιότητα μιας λύσης.

Ο αλγόριθμος PSO είναι **μεταευρετικός**[1] (*metaheuristics*) και επομένως, δεν μπορεί να εγγυηθεί την υπάρξη μιας βέλτιστης λύσης ενός προβλήματος βελτιστοποίησης. Ο αλγόριθμος επιλύει προβλήματα βελτιστοποίησης συνεχών (*continuous*) και διακριτών (*discrete*) μη-γραμμικών συναρτήσεων.

[1] Η διαφορά ανάμεσα στην ευρετική (*heuristics*) και την μεταευρετική (*metaheuristics*) είναι ότι η πρώτη χρησιμοποιεί απλές μεθόδους και τεχνικές για την εύρεση μιας λύσης ενός προβλήματος που δεν μπορεί να επιλυθεί αναλυτικά, ενώ η δεύτερη λειτουργεί επαναληπτικά αναζητώντας σε έναν χώρο αναζήτησης μια ολοένα και καλύτερη λύση (βελτιστοποίηση).

2.3 Canonical PSO

Στην κανονική του μορφή, ο αλγόριθμος PSO, όπως επινοήθηκε από τους Eberhart & Kennedy (1995) μεταβάλλει σε κάθε γενιά την κατάσταση κάθε σωματιδίου του σμήνους με την βοήθεια των εξής παραμέτρων:

- την τρέχουσα ταχύτητά του,
- την δική του καλύτερη μέχρι στιγμής θέση (γνωστή ως *pbest*), και
- την καλύτερη μέχρι στιγμής θέση που έχει ανακαλυφθεί από ολόκληρο το σμήνος (γνωστή ως *gbest*).

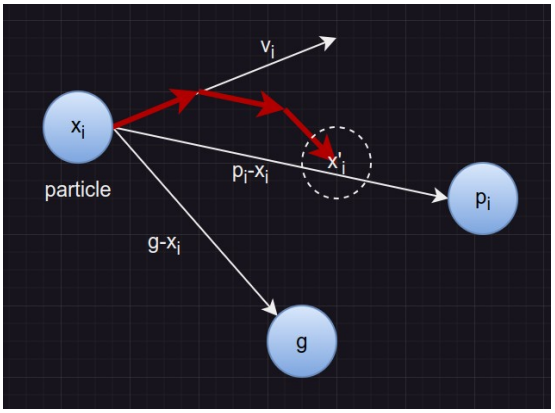
Η διαδικασία εξεύρεσης μίας βέλτιστης λύσης ολοκληρώνεται, όταν επιτευχθεί ένας στόχος ή όταν ολοκληρωθούν οι γενιές.

2.3.1 Μαθηματικό μοντέλο

Έστω ένα σμήνος N σωματιδίων. Τα σωματίδια κινούνται σε ένα χώρο αναζήτησης D διαστάσεων. Ο διανυσματικός χώρος οριοθετείται από ένα κάτω όριο x_j^{low} και ένα άνω όριο x_j^{up} όπου $j = 1, 2, \ldots, D$.

Κάθε σωματίδιο i έχει σε κάθε γενιά t:

- Ένα διάνυσμα θέσης $x_{ij}(t)$
- Ένα διάνυσμα ταχύτητας $v_{ij}(t)$
- Ένα διάνυσμα $p_{ij}(t)$ που αποθηκεύει την καλύτερη θέση (*personal best*) που έχει βρεθεί μέχρι στιγμής από το σωματίδιο i



Σχήμα 1: Σωματίδιο στον χώρο αναζήτησης

- Μία τιμή κόστους $c_i = f(x_{ij})$
- Ένα διάνυσμα $g_{ij}(t)$ που αποθηκεύει την καλύτερη θέση στην γειτονιά του σωματιδίου i (*local best*) ή σε ολόκληρο το σμήνος (*global best*) αναλόγως του μοντέλου.

όπου $f : \mathbb{R}^D \to \mathbb{R}$ είναι μία αντικειμενική συνάρτηση, η οποία πρέπει να ελαχιστοποιηθεί και $i = 1, 2, \ldots, N$.

2.3.2 Ο ρόλος της γειτονιάς

Έστω B_i το σύνολο που περιλαμβάνει το σωματίδιο i και όλους τους γείτονές του. Τα περιεχόμενα του συνόλου B_i εξαρτώνται από την τοπολογία γειτονιάς που έχει επιλεγεί (πλήρως συνεκτική, δακτυλίου, κλπ). Αν κάθε σωματίδιο μπορεί να επικοινωνεί με όλα τα άλλα, τότε

$$B_i = I = \{1, 2, \ldots, N\}, \forall i \in I$$

όπου I είναι το σύνολο όλων των σωματιδίων. Τότε, το διάνυσμα $g_j(t) = g_{ij}(t), \forall i \in I$ αποθηκεύει κάθε χρονική στιγμή την καλύτερη θέση που έχει εντοπιστεί σε ολόκληρο το σμήνος και το μοντέλο λέγεται **Global PSO** (Parsopoulos, 2018).

Στην περίπτωση που η τοπολογία γειτνίασης δεν περιλαμβάνει όλα τα σωματίδια του σμήνους, τότε κάθε σωματίδιο επικοινωνεί μόνον με τα σωματίδια της γειτονιάς του και όχι με όλα, δηλαδή $B_i \subset I$ και το διάνυσμα $g_{ij}(t)$ αποτυπώνει την καλύτερη θέση που έχει βρεθεί στην γειτονιά του σωματιδίου i. Σε αυτή την περίπτωση, το μοντέλο λέγεται **Local PSO** (Parsopoulos, 2018).

2.3.3 Αρχικοποίηση του σμήνους

Η αρχικοποίηση του διανύσματος θέσης κάθε σωματιδίου γίνεται συνήθως επιλέγοντας τυχαίες τιμές στο διάστημα $\left[x_j^{min}, x_j^{max}\right]$ ακολουθώντας ομοιόμορφη κατανομή. Με τον ίδιο τρόπο αρχικοποιείτα και η ταχύτητα κάθε σωματιδίου επιλέγοντας τυχαίες τιμές ομοιόμορφα στο διάστημα $\left[-v_j^{max}, v_j^{max}\right]$ όπου

$$v_j^{max} = \lambda_j \left(x_j^{max} - x_j^{min}\right) \tag{1}$$

και $\lambda \in (0, 1)$. Εναλλακτικά, μπορούμε να θέσουμε τα διανύσματα των ταχυτήτων ίσα με το μηδέν. Το διάνυσμα $p_{ij}(t)$ κάθε σωματιδίου λαμβάνει ως αρχική τιμή την θέση του σωματιδίου. Το διάνυσμα $g_{ij}(t)$ προσδιορίζεται αρχικώς ως η καλύτερη θέση από τις αρχικές θέσεις που έχουμε ήδη.

2.3.4 Επαναλήψεις (γενιές)

Σε κάθε γενιά, η νέα θέση και η νέα ταχύτητα κάθε σωματιδίου υπολογίζονται από τις σχέσεις:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{2}$$

$$\begin{aligned} v_{ij}(t+1) =& v_{ij}(t) + \\ & \varphi_p r_p \left[p_{ij}(t) - x_{ij}(t)\right] + \\ & \varphi_g r_g \left[g_{ij}(t) - x_{ij}(t)\right] \end{aligned} \tag{3}$$

όπου

- φ_p είναι ο **γνωστικός συντελεστής** (*cognitive coefficient*), ο οποίος ποσοτικοποιεί την απόδοση του σωματιδίου σε σχέση με την τελευταία καλύτερη θέση του.
- φ_g είναι ο **κοινωνικός συντελεστής** (*social coefficient*), ο οποίος ποσοτικοποιεί την απόδοση του σωματιδίου σε σχέση με όλο το σμήνος.
- r_p, r_g είναι συντελεστές που λαμβάνουν σε κάθε επανάληψη μία τυχαία τιμή ομοιόμορφα κατανεμημένη στο διάστημα $[0, 1]$.

Οι συντελεστές φ_p, φ_g ονομάζονται **συντελεστές επιτάχυνσης**. Στην σχέση (3), ο πρώτος όρος του αθροίσματος λέγεται **ορμή** (*momentum*) και κρατάει μνήμη της προηγούμενης κινητικής κατάστασης του σωματιδίου. Με τον όρο αυτό αποφεύγεται η ραγδαία μεταβολή της κινητικής κατάστασης του σωματιδίου.

Σε κάθε επανάληψη, η νέα θέση ελέγχεται ως προς τα όρια του διανυσματικού χώρου και αν τα υπερβαίνει, τότε τοποθετείται στο όριο. Ομοίως, αν το νέο διάνυσμα της ταχύτητας υπερβαίνει τα όρια της ταχύτητας, τότε τοποθετείται στο όριο.

Τα διανύσματα $p_{ij}(t)$ και $g_{ij}(t)$ ενημερώνονται σε κάθε γενιά t ως εξής:

$$\mathrm{p}_{ij}(t+1) = \begin{cases} x_{ij}(t+1), & \text{if } f(x_{ij}(t+1)) < f(p_{ij}(t)) \\ p_{ij}(t), & \text{otherwise} \end{cases} \tag{4}$$

$$g_{ij} \in \{p_{kj}, k \in B_i\} : f(g_{ij}) = \min_{k \in B_i} \{f(p_{kj})\} \tag{5}$$

2.3.5 Βάρος Αδράνειας

Στην εξίσωση (3), αν η ορμή είναι πολύ μεγάλη, τότε μπορεί να οδηγήσει σε μία εκρηκτική αύξηση της ταχύτητας και σε αδυναμία σύγκλισης. Προκειμένου να αποφευχθεί κάτι τέτοιο, οι Shi & Eberhart (1998) πρότειναν την εισαγωγή μιας παραμέτρου γνωστής ως **βάρος αδράνειας** (*inertia weight*). Έτσι, η εξίσωση (3) γράφεται ως εξής:

$$\begin{aligned} v_{ij}(t+1) =& wv_{ij}(t)+ \\ & \varphi_p r_p \left[p_{ij}(t) - x_{ij}(t)\right] + \\ & \varphi_g r_g \left[g_{ij}(t) - x_{ij}(t)\right] \end{aligned} \tag{6}$$

όπου w είναι το βάρος αδράνειας. Το βάρος αδράνειας, το οποίο καθορίζει την επίδραση της ορμής του σωματιδίου στην επόμενη θέση του ξεκινά συνήθως με μια μεγάλη τιμή w_{max} και σταδιακά ελαττώνεται σε μια χαμηλή τιμή w_{min} ακολουθώντας την σχέση:

$$w(t) = w_{max} - \frac{t}{t_{max}}(w_{max} - w_{min}) \tag{7}$$

όπου t_{max} είναι το μέγιστο πλήθος γενεών. Ωστόσο, οι Clerc & Kennedy (2002) εισηγήθηκαν ότι για να υπάρχει σύγκλιση του μοντέλου πρέπει το βάρος αδράνειας να λαμβάνει σταθερή τιμή $w = 0.729$ και οι συντελεστές επιτάχυνσης να ισούνται με $\varphi_p = \varphi_g = 1.49$.

2.3.6 Συντελεστές συμπίεσης

Προκειμένου να αποφευχθεί η υπερβολική εξερεύνηση του χώρου αναζήτησης και να υπάρχει καλή σύγκλιση, οι Clerc & Kennedy (2002) πρότειναν οι συντελεστές αδράνειας και επιτάχυνσης, γνωστοί και ως **συντελεστές συμπίεσης** (*constriction coefficients*), να σχετίζονται μεταξύ τους ως εξής:

$$w = \frac{1}{\varphi - 1 + \sqrt{\varphi^2 - 2\varphi}} \tag{8}$$

$$\varphi_p = \varphi_g = \varphi w \tag{9}$$

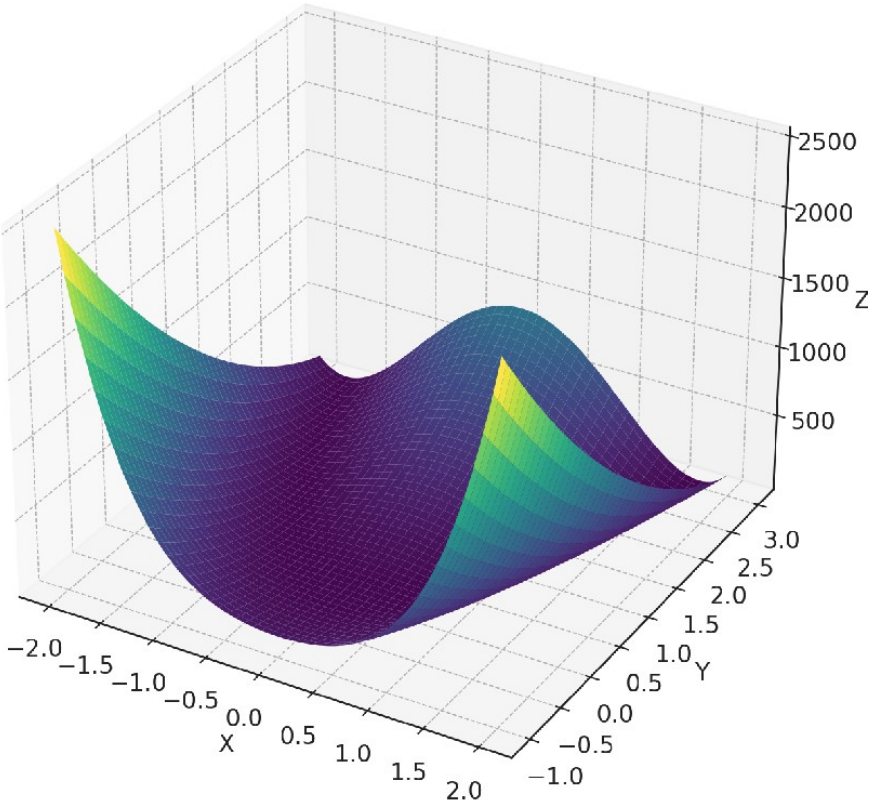
Από την σχέση (8), προκειμένου να είναι η τετραγωνική ρίζα $\sqrt{\varphi^2 - 2\varphi}$ πραγματικός αριθμός και το βάρος αδράνειας $w < 1$, πρέπει η παράμετρος $\varphi > 2$.

2.3.7 Παράδειγμα

Ας πάρουμε για παράδειγμα, την ακόλουθη συνάρτηση, γνωστή ως **συνάρτηση Rosenbrock** (Rosenbrock, 1960):

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Η συγκεκριμένη συνάρτηση (Σχήμα 2) χρησιμοποιείται συχνά ως δοκιμαστικό πρόβλημα για την αξιολόγηση αλγορίθμων βελτιστοποίησης, καθώς γνωρίζουμε πολύ καλά ότι το ολικό ελάχιστο της συνάρτησης βρίσκεται στο σημείο (1,1) και έχει τιμή 0, αλλά είναι πολύ δύσκολο να εντοπιστεί.

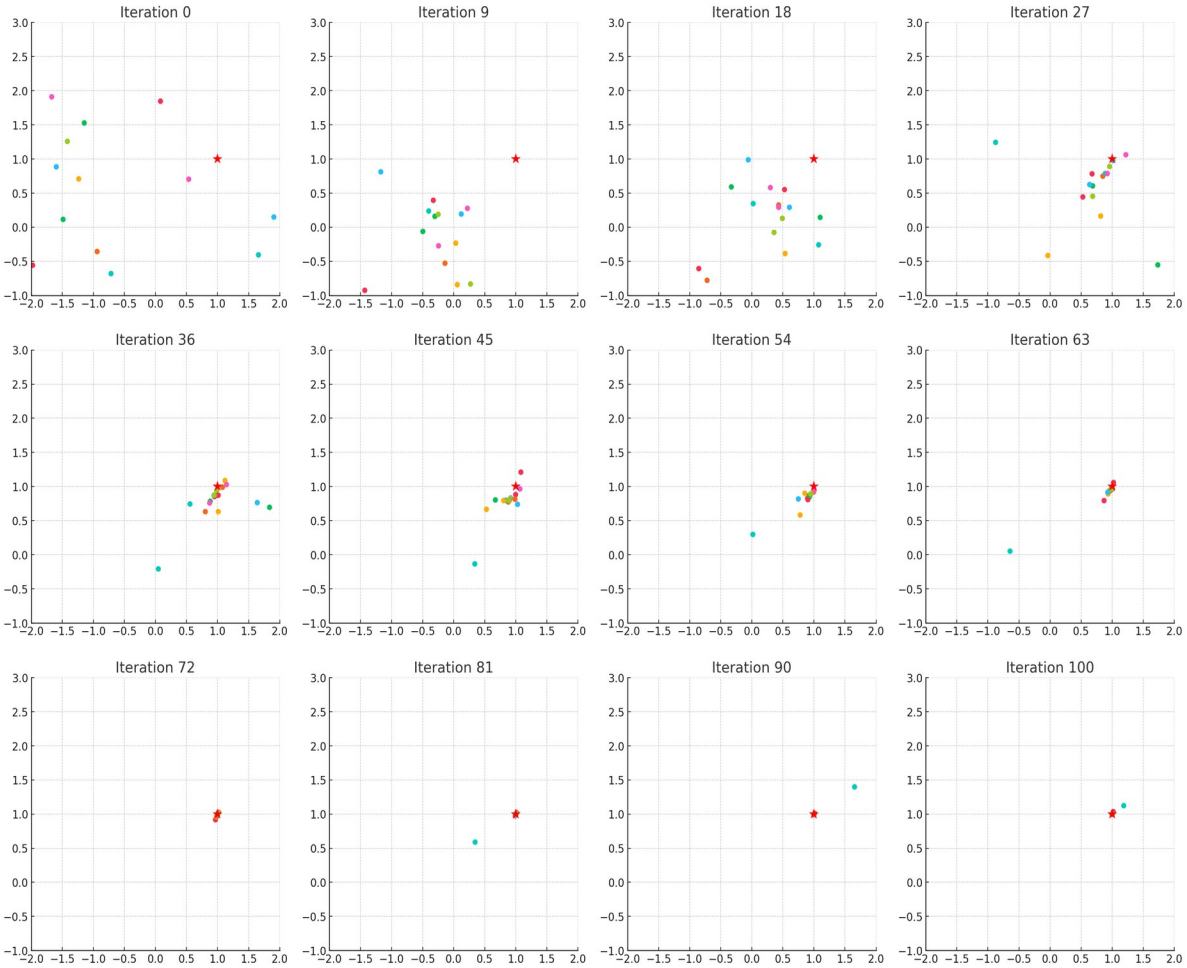


Σχήμα 2: Η συνάρτηση Rosenbrock

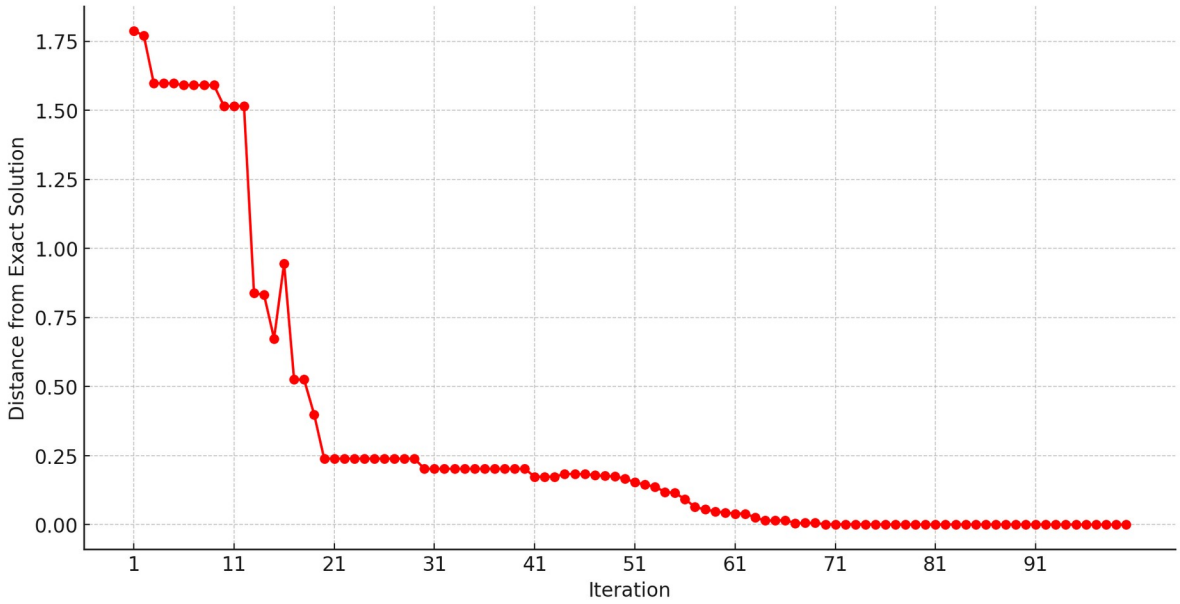
Για την απλή περίπτωση

- 16 σωματιδίων,
- με 100 επαναλήψεις,
- βάρος αδράνειας $w = 0.729$
- γνωστικό συντελεστή $\varphi_p = 1.49$
- κοινωνικό συντελεστή $\varphi_g = 1.49$

ο αλγόριθμος PSO κατάφερε να βρει την λύση πολύ εύκολα. Όπως βλέπουμε στο Σχήμα 3, τα σωματίδια ξεκινούν τυχαία κατανεμημένα στον χώρο αναζήτησης και σιγά σιγά συγκλίνουν προς την λύση (κόκκινο αστέρι). Μάλιστα, ήδη από την επανάληψη 70, ο αλγόριθμος έχει εντοπίσει με επιτυχία την ακριβή λύση (Σχήμα 4).



Σχήμα 3: Γραφική αναπαράσταση της θέσης των σωματιδίων σε επιλεγμένες επαναλήψεις. Με κόκκινο αστέρι δηλώνεται η ακριβής λύση.



Σχήμα 4: Απόσταση του gBest από την ακριβή λύση.

2.4 Binary PSO

Ένα οποιοδήποτε πρόβλημα, συνεχούς ή διακριτού πεδίου, μπορεί να περιγραφεί σε δυαδική μορφή. Για τον λόγο αυτό, οι Kennedy & Eberhart (1997) τροποποίησαν τον αρχικό τους αλγόριθμο έτσι, ώστε να εφαρμόζεται σε διακριτά προβλήματα καταλήγοντας στον αλγόριθμο Binary PSO (BPSO).

Στον αλγόριθμο BPSO, η σχέση μεταβολής της ταχύτητας του σωματιδίου δεν διαφέρει από εκείνη του κανονικού PSO. Η ταχύτητα, όμως αποκτά ένα εντελώς διαφορετικό νόημα. Καθώς το διάνυσμα x_i ενός σωματιδίου i είναι ένας δυαδικός αριθμός, κάθε συντεταγμένη x_{ij} του διανύσματος αυτού έχει τιμή 0 ή 1. Έτσι, η ταχύτητα του σωματιδίου αντιπροσωπεύει το πλήθος των bits της θέσης του σωματιδίου που αλλάζουν σε κάθε επανάληψη (ή διαφορετικά, την απόσταση Hamming ανάμεσα στην θέση του σωματιδίου την χρονική στιγμή t και σε εκείνη την χρονική στιγμή $t+1$). Υπό αυτή την έννοια, η ταχύτητα εκφράζει την πιθανότητα να αλλάξει ένα bit από 0 σε 1. Αν, για παράδειγμα, η ταχύτητα έχει τιμή 0.3 τότε υπάρχει 30% πιθανότητα να αλλάξει το bit σε 1 και 70% να μην αλλάξει.

Συνεπώς, η σχέση (2) που περιγράφει την μεταβολή της θέσης ενός σωματιδίου γίνεται ως εξής:

$$\mathrm{x}_{ij}(t+1) = \begin{cases} 1, & \text{if } rand() < S(v_{ij}(t+1)) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

όπου $rand()$ είναι μία συνάρτηση που παράγει ψευδοτυχαίους αριθμούς με ομοιόμορφη κατανομή στο διάστημα $[0, 1]$ και S είναι μία **συνάρτηση μεταφοράς**. Οι Kennedy & Eberhart (1997) πρότειναν ως συνάρτηση μεταφοράς την σιγμοειδής (*sigmoid*) συνάρτηση:

$$S(v_{ij}(t)) = \frac{1}{1 + e^{-v_{ij}(t)}} \tag{11}$$

Όμως, και άλλες συναρτήσεις μεταφοράς έχουν προταθεί κατά καιρούς (Guo κ.ά., 2020).

2.5 Discrete PSO

Το δυαδικό μοντέλο του αλγορίθμου BPSO αντιμετωπίζει σημαντικές δυσκολίες στην εφαρμογή για την επίλυση προβλημάτων χρονοπρογραμματισμού. Η κύρια πηγή των δυσκολιών έγκειται στην έννοια της ταχύτητας που ενημερώνει το σωματίδιο κάθε χρονική στιγμή. Η περιγραφή της ταχύτητας ως πιθανότητας μεταβολής ενός bit του σωματιδίου δυσχεραίνει την κατάσταση δεδομένου ότι μια ικανοποιητική αναπαράσταση του σωματιδίου στον δυαδικό χώρο είναι πολύ δύσκολο να υλοποιηθεί.

Οι Ribeiro & Urrutia (2004) εξετάζοντας το Traveling Tournament Problem εισήγαγαν για πρώτη φορά την έννοια των **ευρετικών τελεστών** (ή τελεστών γειτνίασης) με τους οποίους θα ασχοληθούμε διεξοδικά στο κεφάλαιο 5.6. Οι ευρετικοί τελεστές, όταν εφαρμοστούν σε ένα ωρολόγιο πρόγραμμα παράγουν ένα νέο ωρολόγιο πρόγραμμα στην γειτονιά του αρχικού.

Λίγο αργότερα, οι Chu κ.ά. (2006) εισήγαγαν μία νέα ερμηνεία της ταχύτητας του σωματιδίου στον αλγόριθμο PSO με την εργασία τους για την επίλυση ενός προβλήματος χρονοπρογραμματισμού σχολικών μαθημάτων (*School Timetabling Problem)*. Η ταχύτητα δανεί-

ζεται χαρακτηριστικά από τους γενετικούς αλγορίθμους. Στην κλασική ερμηνεία της ταχύτητας του σωματιδίου στον αλγόριθμο PSO, όπως φαίνεται στην σχέση (3), ο πρώτος όρος στο δεξί μέλος της ισότητας αντιπροσωπεύει την τρέχουσα ταχύτητα του σωματιδίου. Αυτή η ταχύτητα από μόνη της μπορεί να μεταβάλλει την θέση του σωματιδίου στο κλασικό μοντέλο. Στην θέση της ταχύτητας αυτής, εισάγεται η έννοια της **μετάλλαξης** (*mutation*) του σωματιδίου, η οποία όπως και στην περίπτωση των γονιδίων, μεταλλάσσει με τυχαίο τρόπο ένα μέρος της υπάρχουσας πληροφορίας του σωματιδίου δημιουργώντας ένα νέο σωματίδιο. Οι Chu κ.ά. (2006) όρισαν την μετάλλαξη αυτή ως μια τυχαία ανταλλαγή (*swapping*) δύο στηλών στον πίνακα ωρολογίου προγράμματος, ενέργεια που έχει πάρα πολλά κοινά με τους ευρετικούς τελεστές. Ο δεύτερος όρος στο δεξί μέλος της ισότητας (3), ο οποίος περιέχει τον γνωστικό συντελεστή, συνδέει το σωματίδιο με την μνήμη της καλύτερης προηγούμενης κατάστασής του. Αυτός ο όρος στο νέο μοντέλο αντικαθίσταται από έναν μηχανισμό **διασταύρωσης** (*crossover*), ο οποίος διασταυρώνει πληροφορία του σωματιδίου με το *pbest* του σωματιδίου και παράγει ένα νέο σωματίδιο. Αναλόγως, ο τρίτος όρος στο δεξί μέλος της ισότητας (3), ο οποίος περιέχει τον κοινωνικό συντελεστή, διασταυρώνει πληροφορία προερχόμενη από το *gBest* του σμήνους με το σωματίδιο παράγοντας ένα νέο σωματίδιο.

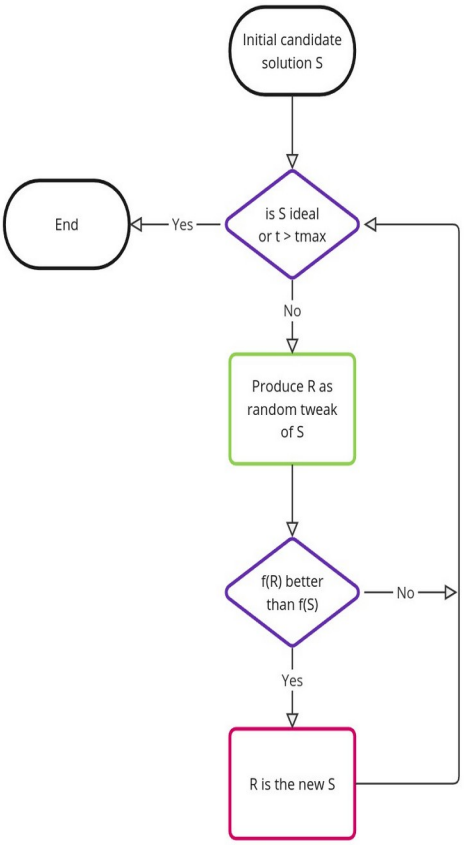
Όπως θα δούμε στο κεφάλαιο 5, η νέα ερμηνεία της ταχύτητας και οι ευρετικοί τελεστές που την υλοποιούν, θα αποτελέσουν τα βασικά εργαλεία στην λειτουργία του υβριδικού αλγορίθμου CP-PSO.

□

3 Βοηθητικοί αλγόριθμοι αναζήτησης και ευρετικές

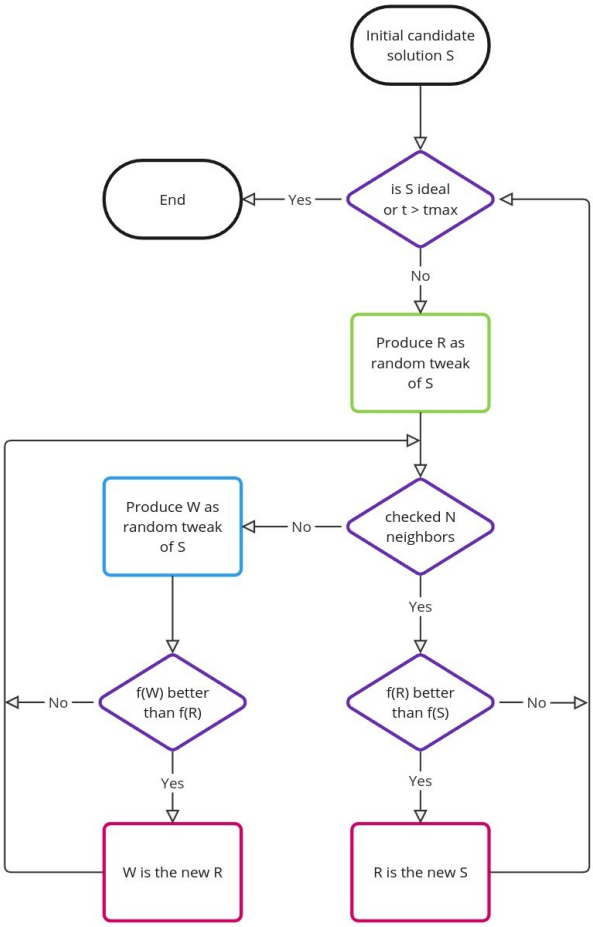
3.1 Hill Climbing

Ο αλγόριθμος Hill Climbing είναι ένας μετευρετικός αλγόριθμος που ανήκει στην οικογένεια των αλγορίθμων τοπικής αναζήτησης (*local search*). Η κεντρική ιδέα του βασίζεται στην διαδικασία που ακολουθεί ένας αναρριχητής που ανεβαίνει έναν λόφο: η κορυφή του λόφου αντιπροσωπεύει την βέλτιστη λύση και κάθε βήμα που κάνει ο αναρριχητής γίνεται προς τα πάνω, δηλαδή προς μία καλύτερη λύση. Ο αλγόριθμος έκανε πρώτη φορά την εμφάνισή του (αν και χωρίς την ονομασία που του δόθηκε αργότερα) στο βιβλίο του Αμερικανού μαθηματικού, George Polya (1945) με τίτλο «*How to Solve It*».



Σχήμα 5: Ο αλγόριθμος Hill Climbing

Ο **αλγόριθμος Hill Climbing** ξεκινά με μία αυθαίρετη λύση του προβλήματος ως αφετηρία και επιχειρεί να την βελτιώσει (Luke, 2013, σ. 15). Με μικρές, *τυχαίες* μεταβολές στην λύση αυτή, παράγει μία γειτονική λύση, η οποία ελέγχεται ως προς την ποιότητά της με ένα μέτρο ποιότητας (αντικειμενική συνάρτηση). Αν η γειτονική λύση είναι καλύτερη, τότε ο αλγόριθμος κάνει «*ένα βήμα προς τα πάνω*» και την αποδέχεται ως νέα αφετηρία. Στην αντίθετη περίπτωση, κάνει μια νέα τυχαία αλλαγή στην αρχική λύση, παράγει μια νέα γειτονική λύση, την εξετάζει ως προς την ποιότητά της και είτε την αποδέχεται ή την απορρίπτει ξανά. Έτσι, με τρόπο επαναληπτικό, κινείται ολοένα και «*ψηλότερα*», έως ότου να φτάσει σε κάποιο σημείο που δεν επιδέχεται περαιτέρω βελτίωση.



Σχήμα 6: Ο αλγόριθμος Steepest Ascent Hill Climbing

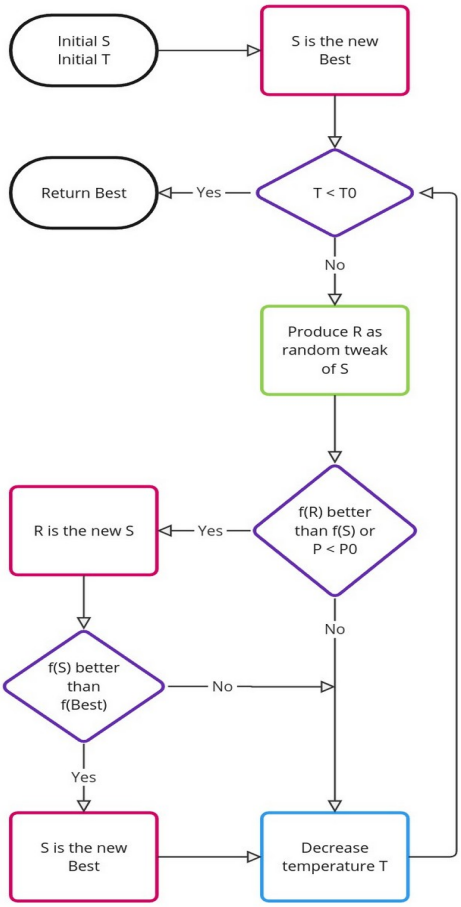
Μια πιο βελτιωμένη παραλλαγή του αλγορίθμου είναι ο αλγόριθμος **Steepest Ascent Hill Climbing** που διαφέρει από τον απλό αλγόριθμο στον ότι δεν εξετάζει μόνον έναν γείτονα σε κάθε επανάληψη, αλλά πολλούς (Luke, 2013, σ. 16). Από τις πολλές γειτονικές λύσεις, κρατάει την καλύτερη, η οποία αν είναι βελτιωτική, την υιοθετεί ως νέα αφετηρία για την επόμενη επανάληψη.

Στο σημείο αυτό πρέπει να κάνουμε μία σημαντική διάκριση ανάμεσα στον αλγόριθμο Hill Climbing και τον αλγόριθμο **Random Search**. Ενώ ο πρώτος κινείται μόνον στην γειτονιά κάθε υποψήφιας λύσης για να παράξει μία νέα, ο δεύτερος επιλέγει εντελώς τυχαία μία νέα λύση που δεν σχετίζεται καθόλου με την αρχική. Με αυτό τον τρόπο, ο αλγόριθμος Random Search εστιάζει μόνον στην **εξερεύνηση** (*exploration*) του χώρου αναζήτησης, ενώ ο αλγόριθμος Hill Climbing στην **εξειδίκευση** (*exploitation*) του.

Ο συνδυασμός των δύο παραπάνω αλγορίθμων παράγει τον αλγόριθμο **Random-Restart Hill Climbing**: ξεκινάμε με μία υποψήφια λύση και κάνουμε Hill Climbing στην γειτονιά της. Αν σε βάθος χρόνου αυτός δεν αποφέρει κάποιο θετικό αποτέλεσμα, τότε επιλέγουμε μία νέα, τυχαία υποψήφια λύση και ξανατρέχουμε τον αλγόριθμο Hill Climbing με αυτήν ως νέα αφετηρία. Με αυτό τον τρόπο, ο αλγόριθμος Random-Restart Hill Climbing επιχειρεί να εξερευνήσει καλύτερα τον χώρο αναζήτησης αποφεύγοντας τυχόν τοπικά βέλτιστα.

3.2 Simulated Annealing

Ο αλγόριθμος Simulated Annealing (SA) είναι ένα πιθανοτικός αλγόριθμος βελτιστοποίησης που αναπτύχθηκε από τους Metropolis κ.ά. (1953) στα πλαίσια του ex-Manhattan Project και εξελίχθηκε από τους Kirkpatrick κ.ά. (1983). Ο αλγόριθμος βασίζεται στην στατιστική μηχανική και πιο συγκεκριμένα στην ανόπτηση των μετάλλων στην μεταλλουργία. Ουσιαστικά, ο SA μιμείται την διαδικασία αργής ψύξης ενός μετάλλου που προηγουμένως έχει θερμανθεί σε υψηλή θερμοκρασία με σκοπό να φτάσει σε μία κατάσταση ελάχιστης ενέργειας. Η ψύξη γίνεται πολύ αργά και η θερμοκρασία πέφτει σιγά σιγά έτσι, ώστε το μέταλλο μεταξύ δύο διαδοχικών διαφορετικών θερμοκρασιών να επιτυγχάνει μια προσωρινή θερμική ισορροπία. Αν το μέταλλο ψυχθεί απότομα, τότε τα άτομά του δεν προλαβαίνουν να καταλάβουν τις ιδανικές θέσεις τους στο κρυσταλλικό πλέγμα, αλλά κατανέμονται με τυχαίο τρόπο που μοιραία οδηγεί σε μία εύθραυστη δομή.



Σχήμα 7: Ο απλός αλγόριθμος SA

Ο αλγόριθμος SA είναι μια εξέλιξη του απλού αλγορίθμου Hill Climbing. Η ειδοποιός διαφορά ανάμεσα στον Hill Climbing και στον SA είναι ότι ο τελευταίος διαθέτει ένα πιθανολογικό κριτήριο αποδοχής λύσεων που τον βοηθούν να δραπευτεύει από τοπικά βέλτιστα. Έτσι, ο SA με τρόπο επαναληπτικό εξερευνά τον χώρο αναζήτησης και αποδέχεται υπό πιθανότητα (που φθείνει με την πάροδο του χρόνου) «*κακές*» λύσεις.

3.2.1 Ο απλός αλγόριθμος

Στην απλούστερη εκδοχή του ο αλγόριθμος SA φαίνεται στο Σχήμα 7. Αρχικά, έχουμε

- μία υποψήφια λύση S που έχει κόστος $f(S)$,
- μία αρχική, υψηλή θερμοκρασία T και
- μία ελάχιστη θερμοκρασία T_0.

Ο αλγόριθμος διατηρεί στην μνήμη του την καλύτερη λύση $Best$ που έχει βρει μέχρι στιγμής, την οποία αρχικοποιούμε με S. Στην συνέχεια, ξεκινούμε μια διαδικασία που επαναλαμβάνεται όσο η θερμοκρασία είναι μεγαλύτερη της ελάχιστης θερμοκρασίας T_0.

Σε κάθε επανάληψη, ο αλγόριθμος επιλέγει τυχαία μια γειτονική λύση R της S. Έτσι,

- Αν η R είναι καλύτερη της S, δηλαδή αν $\Delta f = f(R) - f(S) < 0$, τότε την αποδέχεται αμέσως.
- Αν η R είναι χειρότερη της S, δηλαδή αν $\Delta f > 0$, τότε
  - αποδέχεται την λύση R μόνον αν η πιθανότητα $P < P_0$, όπου $P_0 = e^{-\frac{\Delta f}{kT}}$. Το κριτήριο αυτό λέγεται **κριτήριο Metropolis**. Ή
  - απορρίπτει την λύση R αν η πιθανότητα $P \geq P_0$ και παραμένει εκεί όπου βρίσκεται.

Αν καταφέρει να κάνει ένα βήμα μπροστά, τότε συγκρίνει την νέα λύση R με την λύση $Best$ και την αντικαθιστά με την πρώτη αν αυτή είναι καλύτερη. Στην συνέχεια, ελαττώνει την θερμοκρασία και επαναλαμβάνει την διαδικασία.

Ο απλός αλγόριθμος είναι αποδοτικός, αλλά έχει το σημαντικό μειονέκτημα ότι το σύστημα δεν προλαβαίνει να επιτύχει θερμική ισορροπία μεταξύ δύο διαδοχικών τιμών της θερμοκρασίας, με αποτέλεσμα η εξερεύνηση του χώρου αναζήτησης να είναι πολύ σύντομη.

3.2.2 Ο βελτιωμένος αλγόριθμος

Προκειμένου να βελτιώσουν την απόδοση του απλού αλγορίθμου SA, οι Johnson κ.ά. (1989) εισήγαγαν την έννοια του **μήκους θερμοκρασίας** L υποχρεώνοντας τον αλγόριθμο να εκτελεί L το πλήθος επαναλήψεις μεταξύ δύο διαδοχικών τιμών της θερμοκρασίας (**βήμα θερμοκρασίας**). Ο ψευδοκώδικας του βελτιωμένου αλγορίθμου SA δίνεται στον Πίνακα 1.

```
1   S ← S0
2   Best ← S
3   T ← T1 > 0
4   while T >= T0:
5       i ← 1
6       repeat
7           R ← random neighbor of S
8           D ← f(R) - f(S)
9           if D <= 0 or rand(0, 1) < exp(-D / kT):
10              S ← R
11          if f(S) < f(Best):
12              Best ← S
13          i ← i + 1
14      until i > L
15      T ← r T
16  return Best
```

Πίνακας 1: Ο βελτιωμένος αλγόριθμος SA

Οι βασικές παράμετροι του αλγορίθμου είναι οι εξής:

- η **αρχική θερμοκρασία** T_1 που πρέπει να είναι αρκετά υψηλή, ώστε να δώσει τον χρόνο στον αλγόριθμο να εξελιχθεί και να εξερευνήσει καλύτερα τον χώρο αναζήτησης.
- ο **ρυθμός ψύξης** (*cooling rario)* r, όπου $T' = r \cdot T$ και $0 < r < 1$.
- το **μήκος θερμοκρασίας** (*temperature length*) L, όπως το ορίσαμε παραπάνω.
- το **μήκος της γειτονιάς** (*neighborhood size*) N που ισούται με το συνολικό πλήθος των δυνατών γειτόνων (μεταλλάξεων) μιας λύσης.
- ο **συντελεστής μεγέθους** (*size factor)* λ.
- το **ελάχιστο ποσοστό αποδεκτών λύσεων** (*minimum percentage*) τ: ο αλγόριθμος διατηρεί έναν μετρητή που αυξάνεται κατά 1 κάθε φορά που ολοκληρώνεται ένα βήμα θερμοκρασίας στο οποίο το ποσοστό των αποδεκτών λύσεων είναι μικρότερο ή ίσο του τ. Ο μετρητής μηδενίζεται αν βρεθεί λύση καλύτερη από την αποθηκευμένη στην μνήμη λύση, *Best*. Όταν ο μετρητής αποκτήσει τιμή ίση με 5, τότε ο αλγόριθμος τερματίζεται θεωρώντας πως πλέον δεν υπάρχει βελτίωση.
- ο **συντελεστής αποκοπής** (*cutoff coefficient*) c, ο οποίος βελτιώνει τον χρόνο εκτέλεσης του αλγορίθμου αποκόπτοντας άχρηστες δοκιμές. Δεδομένου ότι αυτό που μετράει είναι το πλήθος των αποδεκτών λύσεων που έχουν βρεθεί σε κάθε βήμα θερμοκρασίας, ο αλγόριθμος αποφασίζει να περάσει στο επόμενο βήμα θερμοκρασί-

ας, όταν το πλήθος των επαναλήψεων γίνει ίσο με L, ή όταν το πλήθος των αποδεκτών λύσεων Q_T στο συγκεκριμένο βήμα θερμοκρασίας ισούται με

$$Q_T = c \cdot L \tag{12}$$

Σύμφωνα με τους Johnson κ.ά. (1989) το μήκος θερμοκρασίας L και το μήκος γειτονιάς N σχετίζονται μεταξύ τους ως εξής:

$$L = \lambda \cdot N \tag{13}$$

Οι παραπάνω παράμετροι παίζουν σημαντικό ρόλο στην απόδοση του αλγορίθμου και η προσεκτική επιλογή των τιμών τους είναι πολύ σημαντική για την επίτευξη αποδοτικών λύσεων.

3.3 Ευρετική fix-and-optimize

Η ευρετική *fix-and-optimize* (Fonseca & Toffolo, 2022) είναι μία προσέγγιση που συνήθως χρησιμοποιείται σε προβλήματα αναζήτησης λύσεων ή βελτιστοποίησης. Η ευρετική είναι σχεδιασμένη να βελτιώνει βηματικά τις παραγώμενες λύσεις και είναι ιδιαίτερα αποδοτική σε πολύπλοκα προβλήματα, όπου η αναζήτηση βέλτιστης λύσης με άμεσο τρόπο ίσως είναι ανέφικτη εξαιτίας υψηλού υπολογιστικού κόστους.

Επιγραμματικά, τα βήματα που ακολουθεί η ευρετική *fix-and-optimize* είναι τα ακόλουθα:

1. *Κατακερματισμός του προβλήματος*: Το πρόβλημα κατακερματίζεται σε επιμέρους, μικρότερα και πιο διαχειρίσιμα προβλήματα.
2. *Σταθερές μεταβλητές*: Σε κάθε επανάληψη, επιλέγεται ένα σύνολο μεταβλητών απόφασης και μετατρέπονται σε σταθερές, δηλαδή η τρέχουσα τιμή τους δεν μεταβάλλεται κατά την διάρκεια της διαδικασίας. Με αυτό τον τρόπο, το πλήθος των βαθμών ελευθερίας του προβλήματος μειώνεται, απλοποιείται η διαδικασία επίλυσης, μειώνεται ο υπολογιστικός χρόνος και ένα μέρος της αρχικής πληροφορίας διατηρείται στην ενδεχόμενη λύση. Η επιλογή των μεταβλητών απόφασης που θα μετατραπούν σε σταθερές στην τρέχουσα επανάληψη γίνεται συνήθως με τυχαίο τρόπο ή μπορεί να ακολουθεί κάποιο κανόνα.

3. *Βελτιστοποίηση*: Μετατρέποντας ένα μέρος από τις μεταβλητές απόφασης σε σταθερές, η διαδικασία βελτιστοποίησης εστιάζει στις υπόλοιπες μεταβλητές του προβλήματος, οι οποίες είναι ελεύθερες να λάβουν μεταβλητές τιμές.
4. *Επαναληπτική διαδικασία*: Η διαδικασία επαναλαμβάνεται για διαφορετικές σταθερές μεταβλητές και αν οδηγήσει σε λύση, αυτή θα αποτελέσει (αναλόγως το είδος του προβλήματος) τα αρχικά δεδομένα της επόμενης επανάληψης. Έτσι, επιτυγχάνεται σταδιακή βελτίωση της αρχικής λύσης.
5. *Σύγκλιση και τερματισμός*: Η ευρετική ολοκληρώνεται όταν επιτευχθεί μία ικανοποιητική λύση με βάση τις απαιτήσεις του προβλήματος.

3.4 Constraint Programming

Ως **Constraint Programming** θεωρείται ένας τομέας στην επιστήμη των υπολογιστών που ασχολείται με προβλήματα που ονομάζονται **Constraint Satisfaction Problems** (CSP), των οποίων η επίλυση απαιτεί την ικανοποίηση ενός συνόλου περιορισμών. Πιο συγκεκριμένα, τα CSP είναι προβλήματα για τα οποία αναζητούμε μία ή περισσότερες λύσεις (ή απλώς την ύπαρξη λύσης) και αποτελούνται από

- Τις μεταβλητές απόφασης (*decision variables*) $x_1, x_2, \ldots, x_n, n \in Z$.
- Τα πεδία ορισμού (*domain*) $d_1, d_2, \ldots, d_n$ κάθε μεταβλητής απόφασης.
- Τους περιορισμούς (*constraints*) $c_1, c_2, \ldots, c_k$, όπου $k \in Z$.

Για την επίλυση των προβλημάτων CSP χρησιμοποιούνται συνήθως οι ακόλουθες μέθοδοι:

3.4.1 Backtracking Search

Η **αναζήτηση με επιστροφή** (*Backtracking Search*) είναι μία μέθοδος αναζήτησης που υλοποιεί τον αλγόριθμο Depth-First-Search (DFS). Η μέθοδος επιχειρεί αυξητικά να δημιουργήσει μία υποψήφια λύση, την οποία απορρίπτει και επιστρέφει στην αμέσως προηγούμενη κατάσταση, αν διαπιστώσει ότι αυτή δεν ικανοποιεί τους περιορισμούς.

Στην αρχή ο αλγόριθμος Backtracking Search επιλέγει μία από τις μεταβλητές του προβλήματος, της αναθέτει μία τιμή και ελέγχει αν ικανοποιούνται οι περιορισμοί. Αν κάποιος πε-

ριορισμός δεν ικανοποιείται, τότε επιστρέφει ένα βήμα πίσω και επιλέγει μία άλλη τιμή της μεταβλητής. Αν όλες οι τιμές της μεταβλητής οδηγούν σε παραβίαση ενός ή περισσοτέρων περιορισμών, τότε επιλέγει μια άλλη μεταβλητή.

Η επιλογή μιας μεταβλητής απόφασης γίνεται είτε με

- **Degree Heuristic**, όπου επιλέγεται η μεταβλητή με την μεγαλύτερη συμμετοχή στους περιορισμούς, είτε με
- **Minimum Remaining Value**, όπου επιλέγεται η μεταβλητή με τις λιγότερες πιθανές τιμές στο πεδίο ορισμού της.

Η επιλογή της τιμής της επιλεγμένης μεταβλητής γίνεται συνήθως με την μέθοδο **Least Constraining Value Heuristic**, όπου η τιμή που επιλέγεται είναι εκείνη που αποκλείει τις λιγότερες τιμές από το πεδίο ορισμού των άλλων μεταβλητών.

3.4.2 Forward Checking

Με την μέθοδο αυτή, ο αλγόριθμος έχοντας αναθέσει μία τιμή σε μία μεταβλητή x του προβλήματος, αφαιρεί από το πεδίο ορισμού των άλλων μεταβλητών που σχετίζονται με την x μέσω των περιορισμών όλες εκείνες τις τιμές που τους παραβιάζουν (**domain pruning**). Έτσι, αποφεύγονται αμέσως τυχόν μελλοντικές ασυνέπειες στις οποίες μπορεί να οδηγήσει η ανάθεση μιας τιμής σε μια μεταβλητή, όπως για παράδειγμα να καταλήξει μια από τις υπόλοιπες μεταβλητές με κενό πεδίο ορισμού.

3.4.3 Constraint Propagation

Η μέθοδος αυτή αποσκοπεί στην μείωση του χώρου αναζήτησης εφαρμόζοντας επαναληπτικά τεχνικές τοπικής συνέπειας (*local consistency)* με πιο γνωστή τον αλγόριθμο **Arc Consistency 3** (AC3).

Επιγραμματικά, ο AC3 λειτουργεί ως εξής: αρχικά δημιουργεί όλα τα τόξα (*arcs*) ανάμεσα σε κάθε μεταβλητή x του προβλήματος με όλες τις άλλες. Ένα τόξο είναι μια μαθηματική έκφραση που περιέχει μόνον την μεταβλητή x στην αριστερή πλευρά της σχέσης και όλες τις υπόλοιπες σχετιζόμενες μέσω των περιορισμών μεταβλητές στην άλλη πλευρά. Αν, για παράδειγμα, έχουμε 3 μεταβλητές A, B, C κάθε μία με πεδίο ορισμού $d = \{1, 2, 3\}$ και 2 πε-

ριορισμούς $A > B$ και $B = C$, τότε το τόξο της μεταβλητής B είναι το $B < A$. Το τόξο $B < A$ είναι *συνεπές*, αν για κάθε τιμή του B, υπάρχει μία τουλάχιστον αποδεκτή τιμή του A. Αν αυτό δεν συμβαίνει, τότε η τιμή του B αφαιρείται από το πεδίο ορισμού της. Ο αλγόριθμος δημιουργεί στην αρχή όλα τα τόξα και τα τοποθετεί σε μία ουρά. Κάθε φορά που το πεδίο ορισμού μιας μεταβλητής αλλάξει, τότε αναζητά όλα τα τόξα που έχουν την μεταβλητή στο αριστερό τους μέλος και τα τοποθετεί στην ουρά. Ο αλγόριθμος ολοκληρώνεται όταν αδειάσει η ουρά.

3.4.4 Backjumping

Όταν κατά την διάρκεια μιας αναζήτησης καταλήξουμε σε αδιέξοδο, η αναζήτηση με επιστροφή μας επιστρέφει στον αμέσως προηγούμενο κόμβο έτσι, ώστε να επιλέξουμε μια άλλη τιμή ή μια άλλη μεταβλητή απόφασης (αν όλες οι τιμές της τρέχουσας μεταβλητής έχουν ήδη δοκιμαστεί). Αντιθέτως, με την μέθοδο Backjumping, κάνουμε ένα «*άλμα προς τα πίσω*» και επιλέγουμε να επιστρέψουμε στον πιο πρόσφατο κόμβο που έχει προκαλέσει κατάργηση τιμής από το πεδίο ορισμού της μεταβλητής στον κόμβο αδιεξόδου.

3.5 SAT

Το πρόβλημα **Boolean Satisfiability** (SAT) είναι ένα πρόβλημα αποφάσεων (*decision problem)* που συγκαταλέγεται στην οικογένεια των **NP-complete** προβλημάτων. Πρόκειται για θεμελιώδες πρόβλημα στην επιστήμη των υπολογιστών και στην μαθηματική λογική.

Το πρόβλημα SAT περιλαμβάνει ένα σύνολο από λογικές μεταβλητές που ικανοποιούν μία λογική σχέση. Η λογική σχέση συνήθως αποτελείται από **προτασιακούς όρους** (*clauses*) που συνδέονται μεταξύ τους με λογικούς τελεστές AND. Κάθε προτασιακός όρος αποτελείται από λογικές εκφράσεις (λογικές μεταβλητές ή την άρνησή τους) συνδεδεμένες με τον λογικό τελεστή OR. Μία ακολουθία από λογικές τιμές για τις μεταβλητές του προβλήματος που ικανοποιούν την λογική σχέση είναι μία λύση του προβλήματος.

Το πρόβλημα SAT είναι ένα πρόβλημα αποφάσεων και επομένως, σκοπός της επίλυσής του δεν είναι να βρούμε όλες τις πιθανές λύσεις, αλλά να αποφανθούμε αν υπάρχει λύση ή όχι.

3.6 CP-SAT

Ο αλγόριθμος ανοικτού κώδικα της οικογένειας εργαλείων βελτιστοποίησης (*OR-Tools*) της Google, CP-SAT είναι ένας υβριδικός αλγόριθμος που συνδυάζει τεχνικές από δύο πεδία: Constraint Programming και Satisfiability Solving. Πρόκειται για έναν αλγόριθμο αναζήτησης που συστηματικά εξερευνά τον χώρο αναζήτησης για πιθανές λύσεις χρησιμοποιώντας μία ποικιλία από εκλεπτυσμένες τεχνικές και ευρετικές αποφάσεων. Με αυτό τον τρόπο, αποκόπτει ανέφικτες διαδρομές οδηγώντας την αναζήτηση σε περιοχές πιο πιθανές για την εύρεση λύσης.

Στο οπλοστάσιο του CP-SAT μπορούμε να βρούμε εξελιγμένες μεθόδους όπως:

- Constraint propagation
- Domain pruning
- Ευρετικές αναζήτησης
- Backtracking search
- Lazy Clause Generation: ο αλγόριθμος αναλύει τις συγκρούσεις που αντιμετωπίζει (δηλαδή καταστάσεις που δεν οδηγούν σε λύση) και δημιουργεί επιπρόσθετους περιορισμούς έτσι, ώστε να αποφευχθεί η μελλοντική επίσκεψη του αλγορίθμου στο ίδιο αδιέξοδο σημείο.
- Cutting plans: σε προβλήματα που μπορούν να μοντελοποιηθούν με τεχνικές Ακέραιου Γραμμικού Προγραμματισμού (*Integer Linear Programming*), ο αλγόριθμος εισάγει νέους, γραμμικούς περιορισμούς (*cutting plans*) που περιορίζουν τον χώρο αναζήτησης σε πιο εφικτές περιοχές.

Ο αλγόριθμος CP-SAT έχει την δυνατότητα να ξεκινά την αναζήτηση από νέα σημεία, όταν διαπιστώνει ότι αυτή έχει τελματώσει και με αυτό τον τρόπο επιτυγχάνει μια ομοιόμορφη αναζήτηση του χώρου λύσεων. Συνάμα, μπορεί να αξιοποιεί το σύνολο της υπολογιστικής ισχύος του συστήματός μας εφαρμόζοντας τεχνικές παράλληλου προγραμματισμού.

3.6.1 Βασικές κλάσεις και μέθοδοι του CP-SAT

Μερικές από τις βασικές κλάσεις και μέθοδοι του CP-SAT που θα χρησιμοποιηθούν αργότερα στην μελέτη μας είναι οι εξής:

1. **Constraint**: κλάση που δημιουργεί ένα αντικείμενο περιορισμού του μοντέλου.

   Μέθοδοι:

   - **OnlyEnforceIf**(*boolvar*): επιβάλλει τον περιορισμό μόνον όταν η λογική παράσταση *boolvar* είναι αληθής.

2. **CpModel**: κλάση που δημιουργεί ένα αντικείμενο μοντέλου.

   Μέθοδοι στιγμιοτύπου:

   - **NewIntVar**(*min*, *max*): δημιουργεί μία ακέραια μεταβλητή με εύρος τιμών στο διάστημα [*min*, *max*].
   - **NewBoolVar**(): δημιουργεί μία λογική μεταβλητή.
   - **Add(***ct***)**: προσθέτει έναν περιορισμό στο μοντέλο. Ο περιορισμός είναι μια γραμμική σχέση που φράσσεται από πάνω ή από κάτω.
   - **AddMaxEquality**(*target*, *variables*): εξετάζει τις μεταβλητές *variables* και αναθέτει την μέγιστη τιμή τους στην μεταβλητή *target*.
   - **AddAbsEquality**(*target*, *variable*): αναθέτει στην μεταβλητή *target* την απόλυτη τιμή της μεταβλητής *variable.*
   - **AddHint**(*var*, *value*): δίνει στο μοντέλο μία ένδειξη τιμής (*value*) για την μεταβλητή *var.*
   - **Minimize**(*objective*): καθορίζει την παράσταση εκείνη του μοντέλου που πρέπει να ελαχιστοποιηθεί.

3. **CpSolver**: κλάση που δημιουργεί ένα αντικείμενο επίλυσης.

   Μεταβλητές στιγμιοτύπου:

   - **parameters.max_time_in_seconds**: καθορίζει ένα ανώτατο όριο χρόνου εκτέλεσης σε δευτερόλεπτα.

- **parameters.num_workers**: καθορίζει τον μέγιστο αριθμό παράλληλων διεργασιών που θα εκκινήσει ο λύτης.
- **parameters.random_seed**: καθορίζει μία τιμή που χρησιμοποιείται για την τυχαιοποίηση του μοντέλου.
- **parameters.log_search_progress**: επιστρέφει αναλυτικές πληροφορίες για την πρόοδο των εργασιών κατά την διάρκεια της εκτέλεσης.

Μέθοδοι στιγμιοτύπου:

- **Solve**(*model*): η μέθοδος που εκκινεί την διαδικασία επίλυσης του μοντέλου. Επιστρέφει μία μεταβλητή κατάστασης (*status*) που λαμβάνει μία από τις ακόλουθες τιμές:
  - **UNKNOWN (0)**: δηλώνει ότι το μοντέλο δεν κατάφερε να βρει λύση πριν λήξει ο προκαθορισμός χρόνος εκτέλεσης, αν αυτός έχει προηγουμένως οριστεί. Στην περίπτωση που δεν έχει οριστεί μέγιστος χρόνος εκτέλεσης, η κατάσταση UNKNOWN δεν επιστρέφεται ποτέ.
  - **MODEL_INVALID (1)**: δηλώνει ότι το μοντέλο δεν πέρασε τον έλεγχο εγκυρότητας (πιθανότατα εξαιτίας κάποιου συντακτικού ή λογικού σφάλματος).
  - **INFEASIBLE (3)**: δηλώνει ότι το μοντέλο απέδειξε ότι το πρόβλημα δεν έχει εφικτή λύση.
  - **FEASIBLE (4)**: δηλώνει ότι βρέθηκε μία εφικτή λύση, αλλά δεν γνωρίζουμε αν είναι βέλτιστη.
  - **OPTIMAL (5)**: δηλώνει ότι βρέθηκε μία βέλτιστη λύση.
- **Value**(): επιστρέφει την τιμή μιας μεταβλητής του μοντέλου μετά την λύση.

Ολόκληρος ο κατάλογος των κλάσεων και των μεθόδων του μοντέλου σε γλώσσα Python βρίσκονται στην επίσημη ιστοσελίδα της Google OR-Tools[2].

□

[2] https://developers.google.com/optimization/reference/python/sat/python/cp_model

4 Πρόβλημα χρονοπρογραμματισμού αγώνων

4.1 Εισαγωγή

Ο χρονοπρογραμματισμός αθλητικών αγώνων (*Sports Timetabling Problem*) είναι ένα συνδυαστικό πρόβλημα βελτιστοποίησης (*combinatorial optimization problem*). Με τον όρο *συνδυαστικό πρόβλημα* εννοούμε ένα πρόβλημα του οποίου η λύση είναι ένας συνδυασμός αντικειμένων πεπερασμένου πλήθους. Στην περίπτωση του χρονοπρογραμματισμού αθλητικών αγώνων, η λύση του προβλήματος είναι ένα **ωρολόγιο πρόγραμμα** (*timetable*), όπου συμμετέχει ένας πεπερασμένος αριθμός ομάδων που αγωνίζονται μεταξύ τους σε ένα πεπερασμένο αριθμό χρονοθυρίδων (*time slots*) ικανοποιώντας ένα πλήθος από κανόνες και περιορισμούς. Το πρόβλημα χρονοπρογραμματισμού αθλητικών αγώνων συγκαταλέγεται στην οικογένεια των NP-hard προβλημάτων.

Σε ακαδημαϊκό επίπεδο, το πρόβλημα χρονοπρογραμματισμού αθλητικών αγώνων απασχόλησε για πρώτη φορά τους Ball & Webster (1977). Έκτοτε, πολλές εργασίες έχουν ασχοληθεί με το θέμα αυτό και τις επιμέρους πτυχές του. Επιγραμματικά, τα προβλήματα χρονοπρογραμματισμού αγώνων χωρίζονται στις ακόλουθες κατηγορίες:

- *Traveling Tournament Problems*: στόχος των προβλημάτων αυτών είναι η εύρεση μίας βέλτιστης λύσης που να ελαχιστοποιεί την συνολική απόσταση που διανύουν οι ομάδες σε ένα τουρνουά. Την εισήγησή τους έκαναν για πρώτη φορά οι Easton κ.ά. (2001).
- *Minimum Break Problems*: αναζητείται μία βέλτιστη λύση που να ελαχιστοποιεί το συνολικό πλήθος των διαλειμμάτων (*breaks*) των συμμετεχόντων ομάδων.
- *Minimum Cost Problems*: αντιστοιχίζοντας κόστος στην εκτέλεση ενός αγώνα, όπως για παράδειγμα το ενοίκιο που πρέπει να πληρώσει μία ομάδα για να παίξει σε ένα γήπεδο που ανήκει σε δημόσιο οργανισμό, αναζητούμε μία βέλτιστη λύση που να ελαχιστοποιεί το συνολικό κόστος των αγώνων σε ένα τουρνουά.
- *Minimum Carryover Effect:* πρόκειται για προβλήματα που αναζητούν μία λύση που ελαχιστοποιεί το φαινόμενο *carryover* μεταξύ των ομάδων (το οποίο θα εξηγήσουμε παρακάτω).
- *Soft Constraints Problems*: πρόκειται για τον πιο σημαντικό τύπο προβλημάτων χρονοπρογραμματισμού αγώνων. Στα προβλήματα αυτά, αναζητούμε μία βέλτιστη λύ-

ση, δηλαδή ένα ωρολόγιο πρόγραμμα, που να ικανοποιεί πλήρως ένα σύνολο από ισχυρούς περιορισμούς και να ελαχιστοποιεί την τιμή μιας αντικειμενικής συνάρτησης που σχετίζεται με το κόστος των ασθενών περιορισμών που παραβιάζονται.

- *Constraint Satisfaction Problems*: ανάλογος τύπος προβλημάτων με της προηγούμενης κατηγορίας, όπου όμως απουσιάζουν εντελώς οι ασθενείς περιορισμοί.

Στην παρούσα εργασία, θα εστιάσουμε σε προβλήματα χρονοπρογραμματισμού τύπου *soft constraints* που είναι και τα προβλήματα του διαγωνισμού ITC2021.

4.2 Ορολογία

Η βασική ορολογία ενός προβλήματος χρονοπρογραμματισμού αγώνων είναι η εξής:

- **Χρονοθυρίδα** (*time slot*): είναι μία περίοδος στον χρόνο (όπως ώρες ή ημέρες) εντός της οποίας μία ομάδα δεν μπορεί να παίξει περισσότερους από έναν αγώνες.
- **Γύρος** (*round*): είναι ένα σύνολο από χρονοθυρίδες κατά την διάρκεια των οποίων μία ομάδα μπορεί να παίξει το πολύ ένα αγώνα.
- **Ωρολόγιο πρόγραμμα** (*timetable*): είναι μία απεικόνιση που απεικονίζει κάθε αγώνα του συνόλου G σε μία χρονοθυρίδα με τέτοιο τρόπο, ώστε μία ομάδα να παίζει το πολύ έναν αγώνα σε μία χρονοθυρίδα,

όπου

- $G = \{(i, j) | \forall i, j \in T, i \neq j\}$ είναι το σύνολο των αγώνων.
- $T = \{0, 1, 2 \ldots, N-1\}$ είναι σύνολο των ομάδων που συμμετέχουν στους αγώνες και ισχύει $N = |T|$
- $S = \{0, 1, 2, \ldots, P-1\}$ είναι το σύνολο των χρονοθυρίδων και $P = |S|$.

Κάθε διατεταγμένο ζεύγος (i, j) συμβολίζει τον αγώνα της ομάδας i που παίζει στην έδρα της με την ομάδα j. Το πλήθος των αγώνων που έχει παίξει η ομάδα i στην έδρα της με την ομάδα j συμβολίζεται με $g_{i,j}$ και ισχύει:

$$g_{i,j} = \sum_{s \in S} g_{i,j,s} \tag{14}$$

όπου

$$\mathrm{g}_{i,j,s} = \begin{cases} 1, \text{if } i \text{ plays a home game with } j \text{ at slot} s \\ 0, \text{otherwise} \end{cases} \tag{15}$$

Από τον ορισμό της χρονοθυρίδας προκύπτει ότι

$$\sum_{j \in T} (g_{i,j,s} + g_{j,i,s}) < 2, \forall i \in T, s \in S \tag{16}$$

Η απαίτηση αυτή χαρακτηρίζεται ως **Κανόνας 0**. Επίσης, μία ομάδα δεν μπορεί να παίζει με τον εαυτό της, δηλαδή:

$$g_{i,i,s} = 0, \forall s \in S \tag{17}$$

που ονομάζεται **Κανόνας 1**.

4.3 Τουρνουά

Τα τουρνουά διακρίνονται σε:

- **k Round-Robin** (kRR): σε αυτό τον τύπο τουρνουά ισχύει ότι:

$$g_{i,j} + g_{j,i} = k, \forall i, j \in T, i \neq j \text{ and } |g_{i,j} - g_{j,i}| \leq 1 \tag{18}$$

- **k Bipartite Round-Robin** ($kBRR$): οι ομάδες χωρίζονται σε δύο σύνολα, V_1, V_2 με $V_1 \cap V_2 = \varnothing$ και

$$\mathrm{g}_{i,j} + g_{j,i} = \begin{cases} k, & \text{if } i \in V_1 \text{and } j \in V_2 \text{ and } |g_{ij} - g_{ji}| \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

- **non Round-Robin** (NRR): κανένα από τα παραπάνω.

Επίσης, τα τουρνουά χωρίζονται σε:

- **Συμπαγή** (*compact*), όπου το πλήθος P των διαθέσιμων χρονοθυρίδων του ωρολογίου προγράμματος είναι το ελάχιστο δυνατό, δηλαδή

$$\mathrm{P} = \begin{cases} k \cdot (N-1), & \text{if } N(mod2) = 0 \\ k \cdot N, & \text{if } N(mod2) = 1 \end{cases} \tag{19}$$

  για ένα τουρνουά τύπου kRR με N ομάδες, και

$$P = k \cdot max(|V_1|, |V_2|) \tag{20}$$

  για ένα τουρνουά τύπου $kBRR$.

- **Time-relaxed**: αν δεν είναι συμπαγή.

4.4 Συμμετρία

Σε ένα τουρνουά τύπου kRR με $k > 1$ η σεζόν μπορεί να χωριστεί σε k διαστήματα, όπου κάθε διάστημα περιέχει ένα ωρολόγιο πρόγραμμα τύπου 1RR και έχει διάρκεια μία σειρά από P/k διαδοχικές χρονοθυρίδες. Σε αυτή την περίπτωση λέμε ότι το ωρολόγιο πρόγραμμα είναι σε **φάσεις** (*phased*).

Ένα ωρολόγιο πρόγραμμα σε φάσεις μπορεί να είναι:

- **Κατοπτρικό** (*mirrored*) αν οι αγωνιζόμενοι στο πρώτο διάστημα είναι ίδιοι με τους αγωνιζόμενους του δεύτερου διαστήματος.
- **Ανεστραμμένο** (*inverted*) αν οι αγώνες ενός διαστήματος παίζονται με την αντίστροφη σειρά σε σχέση με το προηγούμενο διάστημα.
- **Αγγλικού τύπου** (*English*) αν οι αντίπαλοι της πρώτης χρονοθυρίδας ενός διαστήματος είναι ίδιοι με τους αντιπάλους της τελευταίας χρονοθυρίδας του προηγούμενου διαστήματος, και οι αντίπαλοι της λ χρονοθυρίδας του διαστήματος ταυτίζονται με τους αντιπάλους της $\lambda - 1$ χρονοθυρίδας του προηγούμενου διαστήματος.
- **Γαλλικού τύπου** (*French*) αν οι αντίπαλοι της τελευταίας χρονοθυρίδας ενός διαστήματος είναι ίδιοι με τους αντιπάλους της πρώτης χρονοθυρίδας του προηγούμενου διαστήματος, και οι αντίπαλοι της λ χρονοθυρίδας του διαστήματος ταυτίζονται με τους αντιπάλους της $\lambda + 1$ χρονοθυρίδας του προηγούμενου διαστήματος.

4.5 Δικαιοσύνη

Μία ομάδα λέμε ότι βρίσκεται σε

- **Home stand**, αν παίζει πολλά παιχνίδια στην έδρα της, το ένα μετά το άλλο.
- **Road trip**, αν παίζει πολλά παιχνίδια εκτός έδρας, το ένα μετά το άλλο.

Μία ομάδα που δεν αγωνίζεται σε μία χρονοθυρίδα λέμε ότι έχει ένα **bye** (πρόκριση χωρίς αγώνα) στην χρονοθυρίδα αυτή.

Μία ομάδα που παίζει στην έδρα της (εκτός έδρας) και στο προηγούμενο παιχνίδι έχει παίξει πάλι στην έδρα της (εκτός έδρας), ανεξάρτητα των byes που έχουν μεσολαβήσει, λέμε ότι έχει ένα **home (away) break**.

Αν μία ομάδα Α αγωνίζεται με την ομάδα Β και στην επόμενη χρονοθυρίδα αγωνίζεται με την ομάδα Γ, τότε λέμε πως η Β δίνει ένα πλεονέκτημα στην Γ, γνωστό ως **carryover effect** (COE). Η επίδραση αυτού του φαινομένου σε ολόκληρο το ωρολόγιο πρόγραμμα υπολογίζεται από την ποσότητα

$$coe_{value} = \sum_{i \in T} \sum_{j \in T} c_{i,j}^2 \tag{21}$$

όπου $c_{i,j}$ είναι το σύνολο των πλεονεκτημάτων (COE) που δίνει η ομάδα i στην ομάδα j σε ολόκληρο το τουρνουά. Για ένα ωρολόγιο πρόγραμμα τύπου kRR με n ομάδες ισχύει ότι $coe_{value} \geq kN(N-1)$ και τότε το ωρολόγιο πρόγραμμα λέγεται **COE-Balanced**.

4.6 Περιορισμοί

Υπάρχουν δύο τύποι περιορισμών (*constraints*):

- οι **ισχυροί** (*hard*) περιορισμοί, οι οποίοι πρέπει να ικανοποιούνται πάντα.
- οι **ασθενείς** (*soft*) περιορισμοί, για τους οποίους έχουμε την επιλογή να τους παραβιάζουμε πληρώνοντας, όμως μία ποινή (*penalty*).

Αν c είναι ένας περιορισμός, τότε η παραβίαση του περιορισμού αυτού παράγει ένα **διάνυσμα απόκλισης** (*deviation vector*) d_c και μία **ποινή** $p_c = w_c f_c(d_c)$, όπου f_c είναι μία **συνάρτηση κόστους** και w_c είναι το **βάρος** του κάθε περιορισμού.

Οι Van Bulck κ.ά. (2019) πρότειναν τις ακόλουθες συναρτήσεις κόστους:

$$f_c(d_c) = \sum_i d_{c,i} \tag{22}$$

$$f_c(d_c) = (\sum_i d_{c,i})^2 \tag{23}$$

$$f_c(d_c) = \sum_i d_{c,i}^2 \tag{24}$$

$$f_c(d_c) = \min_i d_{c,i} \tag{25}$$

$$f_c(d_c) = \max_i d_{c,i} \tag{26}$$

όπου ο δείκτης i απαριθμεί τις συντεταγμένες του διανύσματος d_c. Οι περιορισμοί χωρίζονται στις παρακάτω κατηγορίες:

4.6.1 Περιορισμοί χωρητικότητας

Οι περιορισμοί χωρητικότητας (*capacity constraints*) έχουν να κάνουν με το πλήθος των αγώνων εντός ή εκτός έδρας μιας ομάδας. Υπάρχουν οι εξής περιορισμοί:

CA1

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ παίζει το ελάχιστο k_{min} και το μέγιστο k_{max} (home, away, γενικά) αγώνες σε ένα σύνολο χρονοθυρίδων $U \subseteq S$. Για την περίπτωση αγώνων εντός εδράς, κάθε παραβίαση αυτού του κανόνα παράγει το ακόλουθο διάνυσμα απόκλισης:

$$d_i = \max\left(k_{min} - \sum_{j \in T_1}\sum_{s \in U} g_{i,j,s}, \sum_{j \in T_1}\sum_{s \in U} g_{i,j,s} - k_{max}, 0\right), \forall i \in T_1 \tag{27}$$

CA2

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ παίζει το ελάχιστο k_{min} και το μέγιστο k_{max} (home, away, γενικά) αγώνες με τις ομάδες του συνόλου $T_2 \subseteq T$ σε ένα σύνολο χρονοθυρίδων $U \subseteq S$. Για την περίπτωση αγώνων εντός έδρας, κάθε παραβίαση αυτού του κανόνα παράγει το ακόλουθο διάνυσμα απόκλισης:

$$d_i = \max\left(k_{min} - \sum_{j \in T_2}\sum_{s \in U} g_{i,j,s}, \sum_{j \in T_2}\sum_{s \in U} g_{i,j,s} - k_{max}, 0\right), \forall i \in T_1 \tag{28}$$

CA3

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ παίζει το ελάχιστο k_{min} και το μέγιστο k_{max} (home, away, γενικά) αγώνες με τις ομάδες του συνόλου $T_2 \subseteq T$ *για κάθε* σειρά από k χρονοθυρίδες ενός

συνόλου χρονοθυρίδων $U \in S$. Αν αυτό παραβιάζεται, τότε το διάνυσμα της απόκλισης (για αγώνες εντός έδρας) ισούται με:

$$d_i = \sum_{l=0}^{U-k} \left(\max(k_{min} - \sum_{j \in T_2} \sum_{s=l}^{l+k-1} g_{i,j,s}, \sum_{j \in T_2} \sum_{s=l}^{l+k-1} g_{i,j,s} - k_{max}, 0) \right), \quad \forall i \in T_1 \qquad (29)$$

Αν ο περιορισμός εφαρμόζεται σε αγώνες εκτός έδρας, τότε στην παραπάνω σχέση το $g_{i,j,s}$ πρέπει να αλλάξει σε $g_{j,i,s}$.

CA4

Ίδιος περιορισμός με τον CA2 με την διαφορά ότι αφορά στο σύνολο όλων των ομάδων του συνόλου $T_1 \subseteq T$. Η παραβίαση δεν παράγει διάνυσμα, αλλά ένα βαθμωτό μέγεθος:

$$d = \max \left(k_{min} - \sum_{i \in T_1} \sum_{j \in T_2} \sum_{s \in U} g_{i,j,s}, \sum_{i \in T_1} \sum_{j \in T_2} \sum_{s \in U} g_{i,j,s} - k_{max}, 0 \right) \qquad (30)$$

Ο περιορισμός CA4 στην παραπάνω μορφή χαρακτηρίζεται ως *Global*. Υπάρχει και μια άλλη μορφή του: η *Every*, όπου για κάθε χρονοθυρίδα $s \in U$ υπολογίζουμε μία απόκλιση και στο τέλος, αθροίζουμε όλες τις αποκλίσεις σε μια συνολική:

$$d = \sum_{s \in U} \left(\max(k_{min} - \sum_{i \in T_1} \sum_{j \in T_2} g_{i,j,s}, \sum_{i \in T_1} \sum_{j \in T_2} g_{i,j,s} - k_{max}, 0) \right) \qquad (31)$$

CA5

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ παίζει εκτός έδρας το ελάχιστο k_{min} και το μέγιστο k_{max} αγώνες με τις ομάδες του συνόλου $T_2 \subseteq T$ *για κάθε* σειρά από εκτός έδρας αγώνες στο σύ-

νολο $U \subseteq S$. Κάθε παραβίαση του συγκεκριμένου περιορισμού παράγει το ακόλουθο διάνυσμα απόκλισης:

$$d_i = \sum_{s_1 \in U} \sum_{s_2 \in U} a_{i,s_1,s_2} \max \left(k_{min} - \sum_{j \in T_2} \sum_{s=s_1}^{s_2} g_{j,i,s}, \sum_{j \in T_2} \sum_{s=s_1}^{s_2} g_{j,i,s} - k_{max}, 0 \right) \tag{32}$$

$\forall i \in T_1$, όπου $s_2 > s_1$ και

$$\mathrm{a}_{i,s_1,s_2} = \begin{cases} 1, & \text{if } i \text{ starts a road trip on } s_1 \text{ and ends it on } s_2 \\ 0, & \text{otherwise} \end{cases}$$

4.6.2 Περιορισμοί αγώνων

Οι περιορισμοί αγώνων (*game constraints*) επιτρέπουν ή απαγορεύουν συγκεκριμένες αναθέσεις ενός αγώνα σε μια χρονοθυρίδα. Υπάρχουν οι εξής κατηγορίες τέτοιων περιορισμών:

GA1

Έστω ένα σύνολο $Q \subseteq G$ από συγκεκριμένους αγώνες. Ο περιορισμός GA1 επιτρέπει το ελάχιστο k_{min} και το μέγιστο k_{max} τέτοιοι αγώνες να πραγματοποιηθούν σε ένα σύνολο χρονοθυρίδων $U \subseteq S$. Κάθε παραβίαση του περιορισμού αυτού παράγει την εξής βαθμωτή ποσότητα απόκλισης:

$$d = \max \left(k_{min} - \sum_{(i,j) \in Q} \sum_{s \in U} g_{i,j,s}, \sum_{(i,j) \in Q} \sum_{s \in U} g_{i,j,s} - k_{max}, 0 \right) \tag{33}$$

GA2

Αν μία ομάδα του συνόλου $T_1 \subseteq T$ παίζει ένα (εντός έδρας, εκτός έδρας, ή απλό) παιχνίδι με μία ομάδα του συνόλου $T_2 \subseteq T$ στο σύνολο χρονοθυρίδων $S_1 \subseteq S$, αλλά καμία ομάδα του $T_3 \subseteq T$ δεν παίζει (εντός έδρας, εκτός έδρας, ή απλό) παιχνίδι με ομάδα του $T_4 \subseteq T$ στο σύνολο χρονοθυρίδων $S_2 \subseteq S$, τότε έχουμε παραβίαση με ποινή 1. Σε κάθε αντίθετη περίπτωση, δεν υπάρχει ποινή. Η απόκλιση σε αυτή την περίπτωση είναι η εξής:

$$d = \max\left(\min(\sum_{i\in T_1}\sum_{j\in T_2}\sum_{s\in S_1} g_{i,j,s}, 1) - \min(\sum_{i\in T_3}\sum_{j\in T_3}\sum_{s\in S_2} g_{i,j,s}, 1), 0\right) \quad (34)$$

4.6.3 Περιορισμοί στα breaks

Οι περιορισμοί του τύπου αυτού (*break constraints*) καθορίζουν την συχνότητα και τον χρόνο (σε χρονοθυρίδες) των breaks. Διακρίνονται στους εξής:

BR1

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ πρέπει να έχει ακριβώς (mode1 = κενό) ή το πολύ (mode1 = LEQ) k home breaks στο σύνολο χρονοθυρίδων $U \subseteq S$. Αν δεν συμβαίνει αυτό, τότε το διάνυσμα απόκλισης (για την περίπτωση του mode1) είναι το εξής:

$$d_i = \left| k - \sum_{s\in U} b_{i,s}h_{i,s} \right|, \forall i \in T_1 \quad (35)$$

όπου

- $\mathrm{b}_{i,s} = \begin{cases} 1, & \text{if } i \text{ has a break on timeslot } s \\ 0, & \text{otherwise} \end{cases}$
- $\mathrm{h}_{i,s} = \begin{cases} 1, & \text{if } i \text{ plays a home game on timeslot } s \\ 0, & \text{otherwise} \end{cases}$

BR2

Το άθροισμα όλων των (**H**ome, **A**way, **H**ome και **A**way) breaks των ομάδων ενός συνόλου $T_1 \subseteq T$ πρέπει να είναι ακριβώς k στο σύνολο χρονοθυρίδων $U \subseteq S$. Για την περίπτωση των home breaks, η απόκλιση για την παραβίαση του συγκεκριμένου περιορισμού είναι:

$$d = \left| k - \sum_{i\in T_1}\sum_{s\in U} b_{i,s}h_{i,s} \right| \quad (36)$$

BR3

Η διαφορά στα home breaks κάθε ζεύγους ομάδων του συνόλου $T_1 \subseteq T$ δεν πρέπει να είναι μεγαλύτερη από k. Διαφορετικά, η απόκλιση για κάθε ζεύγος ισούται με:

$$d_{i,j} = \max\left(\left|\sum_{s\in S} b_{i,s}h_{i,s} - \sum_{s\in S} b_{j,s}h_{j,s}\right| - k, 0\right), \quad \forall i,j \in T_1, i<j \tag{37}$$

BR4

Κάθε ομάδα του συνόλου $T_1 \subseteq T$ πρέπει να έχει ακριβώς k *road trips* στο σύνολο χρονοθυρίδων S, αλλιώς το διάνυσμα απόκλισης είναι το εξής:

$$d_i = \max\left(k - \sum_{s_1\in S}\sum_{s_2\in S} a_{i,s_1,s_2}, 0\right), \quad s_2 > s_1, \forall i \in T_1 \tag{38}$$

4.6.4 Περιορισμοί δικαιοσύνης

Οι περιορισμοί δικαιοσύνης (*fairness constraints*) είναι περιορισμούς που σκοπό έχουν να αυξήσουν την δικαιοσύνη των αγώνων του τουρνουά. Διακρίνονται στις εξής κατηγορίες:

FA1

Για κάθε ομάδα του συνόλου $T_1 \subseteq T$, η διαφορά ανάμεσα στο πλήθος των εντός και των εκτός έδρας αγώνων στην χρονοθυρίδα $s \in U \subseteq S$ δεν πρέπει να υπερβαίνει το k. Αν αυτό παραβιάζεται, τότε επιστρέφεται ένα διάνυσμα απόκλισης με την μεγαλύτερη διαφορά:

$$d_i = \max_{s\in U}\left(\left|\sum_{j\in T_1}\sum_{p=1}^{s}(g_{i,j,p} - g_{j,i,p})\right| - k, 0\right), \quad \forall i \in T_1 \tag{39}$$

FA2

Η διαφορά στα παιχνίδια εντός έδρας κάθε ζεύγους ομάδων του συνόλου $T_1 \subseteq T$ δεν πρέπει να είναι μεγαλύτερη από k μετά από κάθε χρονοθυρίδα του συνόλου $U \subseteq S$. Αν αυτό δεν ισχύει, τότε επιστρέφουμε μία απόκλιση με την μεγαλύτερη διαφορά:

$$d_{ij} = \max_{s \in U} \left(\left| \sum_{t \in T_1} \sum_{p=1}^{s} (g_{i,t,p} - g_{j,t,p}) \right| - k, 0 \right), \quad \forall i, j \in T_1, i < j \tag{40}$$

FA3

Σε κάθε ζεύγος ομάδων $i, j \in T_1 \subseteq T$, η πρώτη ομάδα παίζει στην έδρα της με την δεύτερη και στην συνέχεια (όχι απαραίτητα στην επόμενη χρονοθυρίδα) η δεύτερη πρέπει να παίξει στην έδρα της με την πρώτη. Αν η ομάδα i παίξει δύο διαδοχικά παιχνίδια με την ομάδα j στην έδρα της, τότε ο κανόνας παραβιάζεται και η απόκλιση ισούται με:

$$d_{i,j} = \sum_{k=1}^{g_{ij}+g_{ji}-1} (1 - |h_{i,j,k} - h_{i,j,k+1}|), \quad \forall i, j \in T_1, i < j \tag{41}$$

όπου

$$\mathrm{h}_{i,j,k} = \begin{cases} 1, & \text{if } i \text{ plays a home game with } j \text{ for } k\text{-th time} \\ 0, & \text{otherwise} \end{cases}$$

FA4

Η τιμή του COE ενός συνόλου ομάδων $T_1 \subseteq T$ δεν μπορεί να υπερβαίνει μια μέγιστη τιμή k. Αν αυτό παραβιάζεται, τότε η απόκλιση ισούται με:

$$d = \max \left(\sum_{i \in T_1} \sum_{j \in T_1} w_{i,j} c_{i,j}^2 - k, 0 \right) \tag{42}$$

όπου

- w_{ij} είναι το βάρος για το διατεταγμένο ζεύγος ομάδων (i, j), και
- c_{ij} είναι το σύνολο των πλεονεκτημάτων (COE) που δίνει η ομάδα i στην ομάδα j σε ολόκληρο το τουρνουά.

FA5

Η συνολική απόσταση που έχουν ταξιδέψει όλες οι ομάδες ενός συνόλου $T_1 \subseteq T$ σε ένα σύνολο χρονοθυρίδων $U \subseteq S$ δεν πρέπει να υπερβαίνει μία μέγιστη τιμή k. Αν αυτό παραβιάζεται, τότε η απόκλιση ισούται με:

$$d = \max\left(\sum_{i \in T_1}\sum_{s \in U} e_{i,s} - k, 0\right) \tag{43}$$

όπου

$$\mathrm{e}_{i,s} = \begin{cases} d_s, \text{ if } i \text{ plays on timeslot } s \\ 0, \text{ otherwise} \end{cases}$$

και d_s είναι η απόσταση που πρέπει να ταξιδέψει η ομάδα i από το γήπεδο του προηγούμενου αγώνα της ως το γήπεδο του τρέχοντος αγώνα της.

FA6

Το συνολικό κόστος για όλους τους αγώνες που έχουν παιχτεί σε ένα σύνολο χρονοθυρίδων $U \subseteq S$ δεν πρέπει να υπερβαίνει μία μέγιστη τιμή k. Αν αυτό δεν συμβαίνει, τότε η απόκλιση ισούται με:

$$d = max\left(\sum_{i \in T}\sum_{j \in T}\sum_{s \in U} c_{i,j,s} g_{i,j,s} - k, 0\right) \tag{44}$$

όπου $c_{i,j,s}$ είναι το κόστος του αγώνα της ομάδας i που παίζει με την ομάδα j στην χρονοθυρίδα s.

4.6.5 Περιορισμοί διαχωρισμού

Οι περιορισμοί διαχωρισμού (*seperation constraints*) αφορούν στον αριθμό των χρονοθυρίδων μεταξύ διαδοχικών αγώνων των ιδίων ομάδων. Διακρίνονται στις ακόλουθες κατηγορίες:

SE1

Για κάθε ζεύγος ομάδων $(i, j) \in T_1 \subseteq T$ πρέπει να μεσολαβούν τουλάχιστον k_{min} χρονοθυρίδες μεταξύ δύο διαδοχικών αμοιβαίων αγώνων για όλες τις χρονοθυρίδες. Αν δεν συμβαίνει αυτό, τότε η απόκλιση ισούται με:

$$d_{i,j} = \sum_{s_1 \in S} \sum_{s_2 = s_1 + 1}^{S} y_{i,j,s_1,s_2} \max\left(k_{min} - (s_2 - s_1 - 1), 0\right), \quad \forall i, j \in T_1, i < j \tag{45}$$

όπου

$$\mathrm{y}_{i,j,s_1,s_2} = \begin{cases} 1, & \text{if } i, j \text{ meet at } s_1 \text{ and then at } s_2 \text{ and not in between} \\ 0, & \text{otherwise} \end{cases}$$

Σε μια πιο γενικευμένη του μορφή, ο περιορισμός αυτός ορίζεται ως εξής: Σε κάθε ζεύγος ομάδων $(i, j) \in T_1 \subseteq T$ πρέπει να μεσολαβούν τουλάχιστον k_{min} χρονοθυρίδες και το πολύ k_{max} χρονοθυρίδες μεταξύ δύο διαδοχικών αμοιβαίων αγώνων για όλες τις χρονοθυρίδες. Αν αυτό δεν συμβαίνει, τότε η απόκλιση ισούται με:

$$d_{i,j} = \sum_{s_1 \in S} \sum_{s_2 = s_1 + 1}^{P} y_{i,j,s_1,s_2} \max(k_{min} - (s_2 - s_1 - 1), (s_2 - s_1 - 1) - k_{max}, 0) \tag{46}$$

$\forall i, j \in T_1, i < j$. Η απαίτηση $i < j$ είναι απαραίτητη, ώστε η συνάρτηση y_{i,j,s_1,s_2} να μην υπολογίζεται διπλά.

SE2

Έστω το σύνολο $Q = \{\{s_1, s_2\}, \{s_3, s_4\}, \ldots\}$ που απαρτίζεται από ζεύγη χρονοθυρίδων. Ο περιορισμός *SE2* μάς λέει πως, αν δύο ομάδες του συνόλου $T_1 \subseteq T$ παίζουν μεταξύ τους στην χρονοθυρίδα s_m, τότε πρέπει να παίζουν μεταξύ τους και στην χρονοθυρίδα s_{m+1}, όπου $\{s_m, s_{m+1}\} \in Q$. Για κάθε ζεύγος χρονοθυρίδων του Q που αυτό παραβιάζεται, παράγεται απόκλιση ίση με:

$$d_{s_1,s_2} = \sum_{\{i,j\} \in T_1} \left| (g_{i,j,s_1} + g_{j,i,s_1}) - (g_{i,j,s_2} + g_{j,i,s_2}) \right|, \quad \forall \{s_1, s_2\} \in Q \tag{47}$$

4.7 Αντικειμενική συνάρτηση

Στα προβλήματα χρονοπρογραμματισμού του διαγωνισμού ITC2021 που εξετάζουμε στην παρούσα εργασία, η συνάρτηση κόστους f_c που χρησιμοποιείται είναι αυτή που δίνεται από την σχέση (22), ενώ το βάρος w_c κάθε περιορισμού αναφέρεται ως «*penalty*» στο αρχείο XML κάθε προβλήματος.

Η αντικειμενική συνάρτηση ενός προβλήματος χρονοπρογραμματισμού του διαγωνισμού ITC2021 είναι η εξής:

$$f_{obj} = \min \left(\sum_{c \in C} w_c \cdot f_c(d_c) \right) \tag{48}$$

όπου

- $c \in C$ είναι ένας περιορισμός του προβλήματος και
- C είναι το σύνολο των όλων των περιορισμών.

Η παραπάνω αντικειμενική συνάρτηση εκφράζει το ελάχιστο κόστος παραβίασης των ασθενών περιορισμών του προβλήματος, δεδομένου ότι μία λύση πρέπει να είναι ωρολόγιο πρόγραμμα που να ικανοποιεί όλους τους ισχυρούς περιορισμούς.

4.8 Σημειογραφία RobinX

Οι Van Bulck κ.ά. (2019) ανέπτυξαν μια σημειογραφία για την αποδοτική κατάταξη των προβλημάτων χρονοπρογραμματισμού αθλητικών αγώνων. Η σημειογραφία περιγράφει ένα πρόβλημα χρονοπρογραμματισμού με την βοήθεια τριών πεδίων (Πίνακας 2):

4.8.1 Πεδίο α

Αποτελείται από 3 υποπαραμέτρους:

- α_1 : τον τύπο του τουρνουά,
- α_2 : την συμπάγεια (*compactness*) του τουρνουά
- α_3 : τον τύπο συμμετρίας

Για παράδειγμα, στο πρόβλημα 2RR, C, ∅ | BR1, CA1, CA2, CA3, GA1, SE2 | ∅, το τουρνουά είναι 2RR, συμπαγές και δεν έχει συμμετρία.

4.8.2 Πεδίο β

Το πεδίο β αφορά στους ενεργούς περιορισμούς (ισχυρούς και ασθενείς) του προβλήματος χρονοπρογραμματισμού αγώνων.

4.8.3 Πεδίο γ

Το πεδίο γ καθορίζει τον τρόπο υπολογισμού της βέλτιστης τιμής.

Για την περίπτωση που υπάρχουν μόνον ισχυροί περιορισμοί, τότε:

- Αν $\gamma = \varnothing$, τότε δεν καθορίζεται τρόπος υπολογισμού βέλτιστης τιμής και το ωρολόγιο πρόγραμμα πρέπει μόνον να ικανοποιεί τους ισχυρούς περιορισμούς.
- Αν $\gamma = BR$, τότε η αντικειμενική συνάρτηση πρέπει να ελαχιστοποιεί το συνολικό πλήθος των breaks.
- Αν $\gamma = TR$, τότε η αντικειμενική συνάρτηση πρέπει να ελαχιστοποιεί την συνολική απόσταση που έχουν ταξιδέψει όλες οι ομάδες που συμμετέχουν στο τουρνουά.

- Αν $\gamma = CR$, τότε στόχος είναι η ελαχιστοποίηση του συνολικού κόστους (όπως αυτό καθορίζεται από αυτό που χρειάζεται να πληρώσει μία ομάδα για να συμμετέχει στο τουρνουά, κ.αλ.)
- Αν $\gamma = CO$, τότε πρέπει η τιμή coe_{value} να είναι ελάχιστη.

Αν υπάρχουν και ασθενείς περιορισμοί, τότε:

- Αν $\gamma = SC$, τότε πρέπει να ελαχιστοποιείται το σύνολο των ποινών για την παραβίαση ενός ή περισσοτέρων ασθενών περιορισμών.

α	α_1	kRR, kBRR, nRR
	α_2	C,R
	α_3	∅, P, M, I, E, F
β	β_1	CA1, CA2, CA3, CA4, CA5
	β_2	GA1, GA2
	β_3	BR1, BR2, BR3, BR4
	β_4	FA1, FA2, FA3, FA4, FA5, FA6
	β_5	SE1, SE2
γ	γ_1	∅, BR, TR, CR, CO, SC

Πίνακας 2: Σημειογραφία RobinX

□

5 Μαθηματικό μοντέλο

5.1 Ορισμός σωματιδίου

Η λύση ενός προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων είναι ένα ωρολόγιο πρόγραμμα που ικανοποιεί υποχρεωτικά όλους τους ισχυρούς περιορισμούς του προβλήματος. Η βέλτιστη λύση είναι εκείνη που ικανοποιεί και τους περισσότερους ασθενείς περιορισμούς.

Στο μαθηματικό μοντέλο που χρησιμοποιούμε, η δομική μονάδα είναι το σωματίδιο. Κάθε σωματίδιο κινείται σε έναν χώρο αναζήτησης πολλών διαστάσεων που καθορίζονται από το πρόβλημα. Η θέση του σωματιδίου στον χώρο αναζήτησης αντιπροσωπεύει ένα ωρολόγιο πρόγραμμα. Αναλόγως με τους περιορισμούς του προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων που ικανοποιεί το συγκεκριμένο ωρολόγιο πρόγραμμα, το σωματίδιο μπορεί να είναι λύση του προβλήματος αυτού ή όχι.

Προτού αναφερθούμε στους τύπους των σωματιδίων, είναι σημαντικό να ορίσουμε τους διαφορετικούς τρόπους κωδικοποίησης της θέσης ενός σωματιδίου στον χώρο αναζήτησης. Οι τρόποι αυτοί είναι οι εξής:

- **κωδικοποίηση** $g_{i,j,s}$:

$$\mathrm{g}_{i,j,s} = \begin{cases} 1, \text{if } i \text{ plays a home game with } j \text{ at slot} s \\ 0, \text{otherwise} \end{cases} \tag{49}$$

- **κωδικοποίηση** $x_{i,s}$:

$$\mathrm{x}_{i,s} = \begin{cases} j, \text{ if } i \text{ plays a home game with } j \text{ on time slot } s \\ -1, \text{ otherwise} \end{cases} \tag{50}$$

  $\forall i \in U, s \in P$

- **κωδικοποίηση** $x_{i,j}$:

$$\mathrm{x}_{i,j} = \begin{cases} s, \text{ if } i \text{ plays a home game with } j \text{ at slot} s \text{ and } i \neq j \\ -1, \text{ otherwise} \end{cases} \tag{51}$$

- **κωδικοποίηση** x_s : κάθε στοιχείο x_s του ωρολογίου προγράμματος X είναι ένα σύνολο από αγώνες $(i, j) \in G$ που εκτελούνται στην χρονοθυρίδα s. Δηλαδή

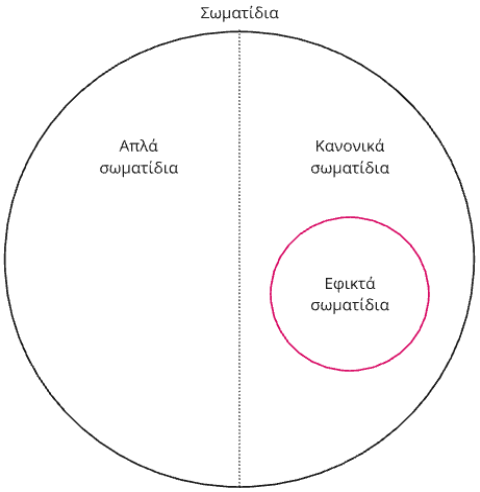
$$x_s = \{(i,j) \in G \mid i,j \in T, i \neq j : i \text{ plays a home game with } j \text{ at slot } s\} \tag{52}$$

5.2 Τύποι σωματιδίων

Όπως είδαμε στο προηγούμενο κεφάλαιο, ένα ωρολόγιο πρόγραμμα αθλητικών αγώνων πρέπει να ικανοποιεί εξ ορισμού τους παρακάτω κανόνες:

- Τον κανόνα 0 (σχέση 16), και
- Τον κανόνα 1 (σχέση 17)

Επιπλέον, για τα προβλήματα χρονοπρογραμματισμού του διαγωνισμού ITC2021 που εξετάζουμε, τα ωρολόγια προγράμματα αθλητικών αγώνων είναι



Σχήμα 8: Τύποι σωματιδίων

- *2-Round-Robin* (2RR), δηλαδή πρέπει να ικανοποιούν την σχέση (18) για $k = 2$.
- Συμπαγή, δηλαδή πρέπει να ικανοποιούν την σχέση (19), και
- Σε φάσεις ή όχι, αναλόγως το πρόβλημα.

Όλοι οι παραπάνω περιορισμοί ονομάζονται **δομικοί (*structural)*** περιορισμοί του προβλήματος.

Με βάση τους τύπους περιορισμών που ικανοποιεί η θέση τους, τα σωματίδια διακρίνονται στις εξής κατηγορίες:

1. **Κανονικά** (*canonical*): πρόκειται για σωματίδια των οποίων η θέση ικανοποιεί τους δομικούς τους περιορισμούς.
2. **Εφικτά** (*feasible*): είναι κανονικά σωματίδια των οποίων η θέση ικανοποιεί τους ισχυρούς τους περιορισμούς.
3. **Απλά** (*simple*): είναι μη-κανονικά σωματίδια.

Συνεπώς, λύσεις ενός προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων είναι μόνον τα εφικτά σωματίδια, διότι μόνον αυτών η θέση ικανοποιεί τους δομικούς και ισχυρούς περιορισμούς του προβλήματος. Έτσι, από το σύνολο των εφικτών σωματιδίων, σκοπός μας είναι η εύρεση μίας βέλτιστης λύσης (εφόσον υπάρχει), δηλαδή ενός σωματίδιου η θέση του

οποίου στον χώρο αναζήτησης ικανοποιεί τους δομικούς και ισχυρούς περιορισμούς και ελαχιστοποιεί το κόστος παραβίασης των ασθενών περιορισμών του προβλήματος.

Καθώς οι περιορισμοί του προβλήματος αφορούν στο ωρολόγιο πρόγραμμα, το οποίο με την σειρά του αντιπροσωπεύεται από την θέση του σωματίδιου, σε ό,τι ακολουθεί θα θεωρούμε χάριν συντομίας ως «περιορισμούς» ενός σωματιδίου τους περιορισμούς που αφορούν στην θέση του.

5.3 Υποσωματίδια

Έστω ένα κανονικό σωματίδιο A ενός προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων T. Έστω, επίσης ότι το σωματίδιο A την χρονική στιγμή t έχει θέση $\overrightarrow{x}$ στον χώρο αναζήτησης.

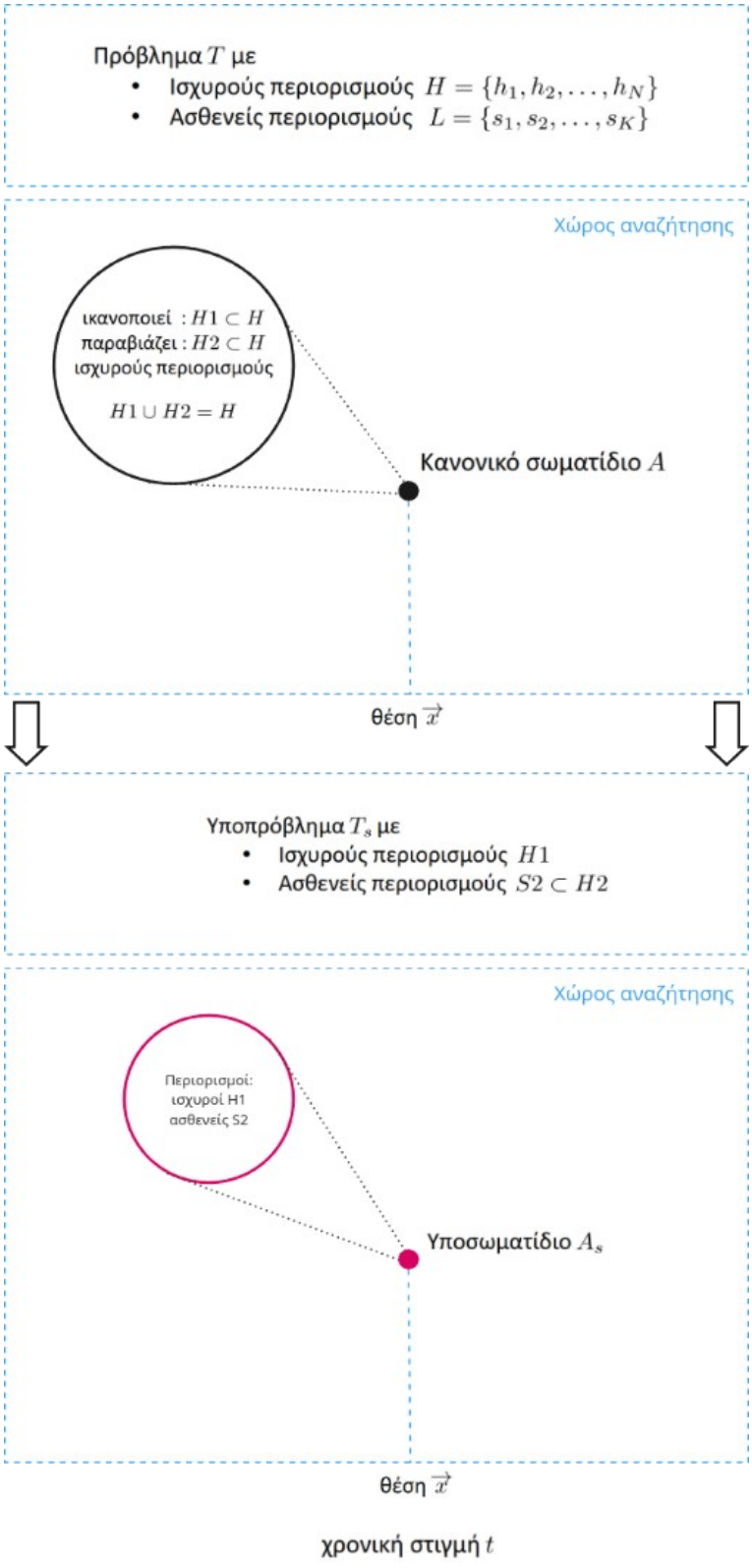
Ως κανονικό σωματίδιο, το A ικανοποιεί τους δομικούς περιορισμούς του προβλήματος T, αλλά όχι όλους τους ισχυρούς περιορισμούς του. Αν H_1 είναι το σύνολο των ισχυρών περιορισμών που ικανοποιεί και H_2 το σύνολο των ισχυρών περιορισμών που παραβιάζει, τότε μπορούμε να ορίσουμε ένα υποπρόβλημα T_s, το οποίο να έχει:

- ως ισχυρούς περιορισμούς όλους τους ισχυρούς περιορισμούς του συνόλου H_1, και
- ως ασθενείς περιορισμούς όλους ή μέρος των ισχυρών περιορισμών του συνόλου H_2

Ένα σωματίδιο A_s του υποπροβλήματος T_s με την ίδια θέση $\overrightarrow{x}$ στον χώρο αναζήτησης την χρονική στιγμή t λέγεται **υποσωματίδιο** (*sub-particle*) του A (Σχήμα 9). Αν το σύνολο S_2 των ασθενών περιορισμών του υποσωματιδίου A_s είναι γνήσιο υποσύνολο του συνόλου H_2, τότε το A_s λέγεται **γνήσιο υποσωματίδιο** (*proper sub-particle*) του A.

Είναι προφανές ότι:

1. Το υποσωματίδιο A_s είναι λύση του υποπροβλήματος T_s και επομένως, είναι εφικτό στο υποπρόβλημα T_s.
2. Από ένα σωματίδιο A μπορούν να προκύψουν ένα ή περισσότερα υποσωματίδια A_s αναλόγως με το πλήθος των ισχυρών περιορισμών του συνόλου H_2 που θα υιοθετηθούν ως ασθενείς περιορισμοί στο υποσωματίδιο.



Σχήμα 9: Παραγωγή υποσωματιδίου από κανονικό σωματίδιο.

Με την βοήθεια των υποσωματιδίων, από ένα κανονικό σωματίδιο A ενός προβλήματος T είναι δυνατόν να παραχθεί ένα εφικτό σωματίδιο (και επομένως, μία λύση του T).

Για παράδειγμα, αν το κανονικό σωματίδιο A ικανοποιεί $H_1 = 15$ ισχυρούς περιορισμούς και παραβιάζει $H_2 = 5$ ισχυρούς περιορισμούς του προβλήματος T, τότε μπορούμε να δημιουργήσουμε ένα υποσωματίδιο A_s ενός υποπροβλήματος T_s με ισχυρούς περιορισμούς όλους τους περιορισμούς του συνόλου H_1, και με ασθενείς περιορισμούς 1 από τους 5 ισχυρούς περιορισμούς του συνόλου H_2 που παραβιάζονται. Βελτιώνοντας το υποσωματίδιο A_s έτσι, ώστε να ικανοποιεί πλήρως τους ασθενείς περιορισμούς του, έχουμε καταφέρει να παράξουμε ένα ωρολόγιο πρόγραμμα που στο πρόβλημα T ικανοποιεί $H_1 = 15 + 1 = 16$ ισχυρούς περιορισμούς και παραβιάζει $H_2 = 5 - 1 = 4$ ισχυρούς περιορισμούς. Από αυτό το νέο σωματίδιο, παράγουμε στην συνέχεια ένα νέο υποσωματίδιο και επαναλαμβάνουμε την διαδικασία έως ότου να ικανοποιούνται και οι 20 ισχυροί περιορισμοί.

5.4 Μέτρο σύγκρισης σωματιδίων

Έστω ένα σωματίδιο Α. Συμβολίζουμε με

- f_{st}^A το κόστος παραβίασης των δομικών περιορισμών του σωματιδίου Α,
- f_h^A το κόστος παραβίασης των ισχυρών περιορισμών του σωματιδίου Α,
- f_s^A το κόστος παραβίασης των ασθενών περιορισμών του σωματιδίου Α.

Ισχύει πάντοτε ότι $f_{st}^A \geq 0, f_h^A \geq 0, f_s^A \geq 0$. Αν f^A είναι το συνολικό κόστος του σωματιδίου Α, τότε:

$$f^A = f_{st}^A + f_h^A + f_s^A \tag{53}$$

Έτσι, αν το σωματίδιο Α είναι

- κανονικό, τότε $f_{st}^A = 0$.
- εφικτό, τότε $f_{st}^A + f_h^A = 0$.

Έστω δύο σωματίδια Α και Β. Συμβολίζουμε με $A \prec B$ την κατάσταση εκείνη κατά την οποία το σωματίδιο Α είναι «καλύτερο» από το σωματίδιο Β και την ορίζουμε ως εξής:

$$(\mathrm{A} \prec B) \quad \text{if} \quad \begin{cases} f_{st}^A < f_{st}^B \text{ and } f_h^A > 0, f_h^B > 0, \text{ or} \\ f_h^A < f_h^B \text{ and } f_{st}^A + f_{st}^B = 0, \text{ or} \\ f_s^A < f_s^B \text{ and } f_{st}^A + f_{st}^B + f_h^A + f_h^B = 0 \end{cases} \tag{54}$$

Δηλαδή, αν τα σωματίδια A και B

- είναι απλά, τότε μέτρο σύγκρισης είναι το κόστος των δομικών περιορισμών.
- είναι κανονικά αλλά όχι εφικτά, τότε μέτρο σύγκρισης είναι το κόστος των ισχυρών περιορισμών.
- είναι εφικτά, τότε μέτρο σύγκρισης είναι το κόστος των ασθενών περιορισμών.

Με άλλα λόγια, ισχύει ότι:

$$\text{feasible particle} \prec \text{canonical particle} \prec \text{simple particle}$$

Με ανάλογο τρόπο ορίζεται η σχέση $(A \succ B)$.

Δύο σωματίδια A, B είναι ίσα (και συμβολίζεται ως $A \equiv B$), αν $g_{i,j,s}^A = g_{i,j,s}^B, \forall (i,j,s)$, δηλαδή αν έχουν το ίδιο ακριβώς ωρολόγιο πρόγραμμα.

5.5 Μετρική

Αν $\Re$ είναι ο χώρος αναζήτησης και $X \in \Re$ είναι ένα σωματίδιο, τότε **γειτονιά** του σωματιδίου X είναι ένα σύνολο $\Re_x \subseteq \Re$ το οποίο αποτελείται από σωματίδια «*πολύ κοντά*» στο X. Το πλήθος των στοιχείων του συνόλου $\Re_x$ λέγεται **μήκος** της γειτονιάς του σωματιδίου X.

Η έννοια της εγγύτητας είναι σχετική και εξαρτάται από το πρόβλημα. Στα προβλήματα χρονοπρογραμματισμού αθλητικών αγώνων που αντιμετωπίζουμε, μπορούμε να την ορίσουμε ως εξής:

Ορισμός: δύο ωρολόγια προγράμματα X, Y απέχουν **μηδενική απόσταση** μεταξύ τους, αν το σύνολο των αγώνων x_s του X στην χρονοθυρίδα s ταυτίζεται με το σύνολο των αγώνων y_s του Y στην χρονοθυρίδα s, για κάθε $s \in S$. Δηλαδή,

$$X = Y \Leftrightarrow \forall s \in S, x_s = y_s$$

Ορισμός: Η απόσταση d δύο ωρολογίων προγραμμάτων X, Y του ιδίου μεγέθους ισούται με:

$$d(X,Y) = \frac{1}{|G|} \sum_{s \in S} \delta(x_s, y_s) \tag{55}$$

όπου

- $\delta(x_s, y_s)$ είναι το πλήθος των διαφορετικών διατεταγμένων ζευγών (i,j) του συνόλου x_s σε σχέση με το σύνολο y_s.
- $|G|$ είναι το συνολικό πλήθος των αγώνων (το οποίο είναι ίδιο και στα δύο ωρολόγια προγράμματα του ιδίου μεγέθους).

Να σημειώσουμε ότι ο όρος $|G|$ διαιρεί το άθροισμα για λόγους κανονικοποίησης έτσι, ώστε η απόσταση d να κανονικοποιείται στο διάστημα $[0,1]$.

Με βάση τα παραπάνω, αν $X, Y \in \Re$ είναι δύο σωματίδια στον χώρο αναζήτησης $\Re$, τότε η συνάρτηση $d : \Re \times \Re \to \mathbb{R}$ είναι μία **μετρική** του χώρου αναζήτησης $\Re$. Ο χώρος αναζήτησης $\Re$ εφοδιασμένος με αυτήν την μετρική ονομάζεται **μετρικός χώρος**. Η μετρική d ικανοποιεί τις παρακάτω ιδιότητες:

- **Μη-αρνητικότητα**: $d(X,Y) \geq 0, \forall X, Y \in \Re$
- **Ταύτιση**: $d(X,Y) = 0$ αν και μόνον εάν $X = Y$
- **Συμμετρία**: $d(X,Y) = d(Y,X), \forall X, Y \in \Re$
- **Τριγωνική ανισότητα**: $d(X,Y) + d(Y,Z) \geq d(X,Z), \forall X, Y, Z \in \Re$

5.6 Ευρετικοί τελεστές

Οι **ευρετικοί τελεστές** (ή αλλιώς, **τελεστές γειτνίασης**) που εισήγαγαν οι Ribeiro & Urrutia (2004) εξελίχθηκαν από τους Anagnostopoulos κ.ά. (2006) και παραλλαγές αυτών χρησιμοποιήθηκαν από διάφορους μελετητές (Rosati κ.ά., 2022) (Dimitsas κ.ά., 2022) για

την επίλυση προβλημάτων χρονοπρογραμματισμού αθλητικών αγώνων. Οι ευρετικοί τελεστές που θα χρησιμοποιήσουμε είναι οι ακόλουθοι:

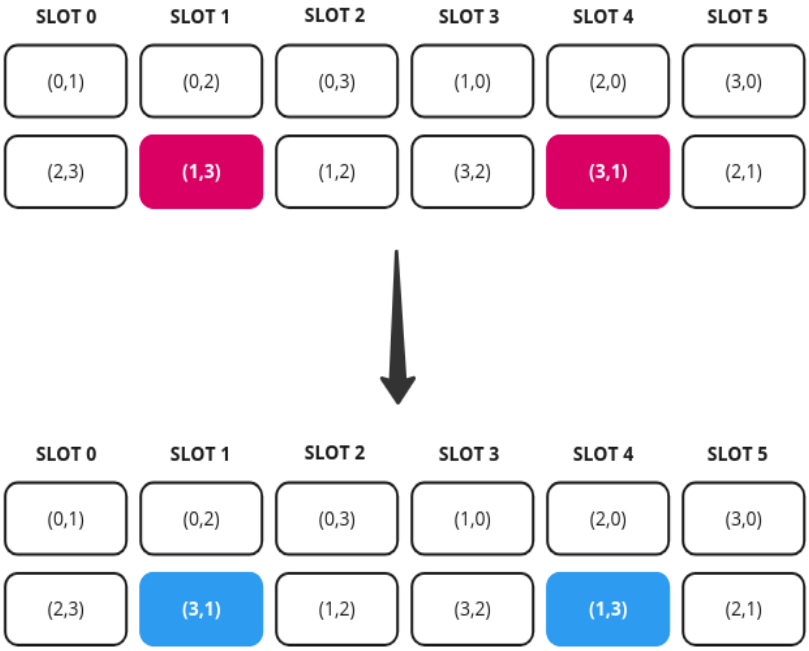
5.6.1 Τελεστής swap_homes

Ο ευρετικός τελεστής *swap_homes* s_h αλλάζει θέση στα αμοιβαία παιχνίδια δύο ομάδων $i, j \in T$ ενός ωρολογίου προγράμματος X και ορίζεται ως εξής:

$$s_h \langle X|i,j\rangle = (X - \{(i,j,s),(j,i,s')\}) \cup \{(j,i,s),(i,j,s')\} \tag{56}$$

Το μήκος της γειτονιάς του X που παράγεται με αυτόν τον τελεστή ισούται με

$$|\Re_x| = \binom{N}{2} \tag{57}$$



Σχήμα 10: Εφαρμογή του τελεστή swap_homes στις ομάδες 1, 3.

όπου N είναι το πλήθος των ομάδων. Στο Σχήμα 10 φαίνεται η σχηματική αναπαράσταση της εφαρμογής του τελεστή *swap_homes* στις ομάδες 1, 3 ενός ωρολογίου προγράμματος αναφοράς.

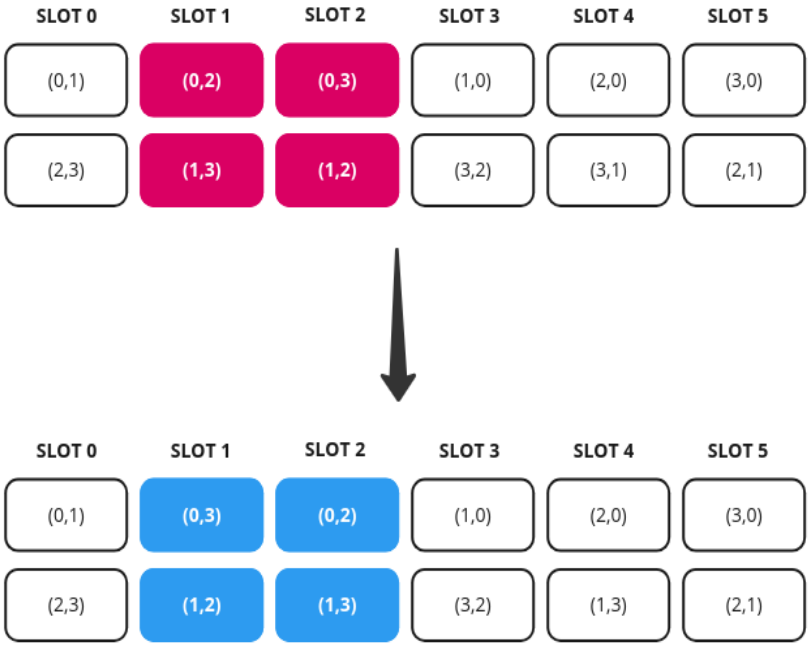
5.6.2 Τελεστής swap_rounds

Ο ευρετικός τελεστής *swap_rounds* s_r αλλάζει όλα τα παιχνίδια μιας χρονοθυρίδας s με όλα τα παιχνίδια μιας διαφορετικής χρονοθυρίδας s' και ορίζεται ως εξής:

$$s_r \langle X|s, s'\rangle = (X - (X_s \cup X_{s'})) \cup (X'_s \cup X'_{s'}) \tag{58}$$

όπου X είναι το ωρολόγιο πρόγραμμα, $s, s' \in S, s \neq s'$ και

- $X_s = \{(i, j, s)|(i, j, s) \in X\}$ είναι τα παιχνίδια στην χρονοθυρίδα s, και
- $X_{s'} = \{(i, j, s')|(i, j, s') \in X\}$ είναι τα παιχνίδια στην χρονοθυρίδα s'



Σχήμα 11: Εφαρμογή του τελεστή swap_rounds στις χρονοθυρίδες 1, 2.

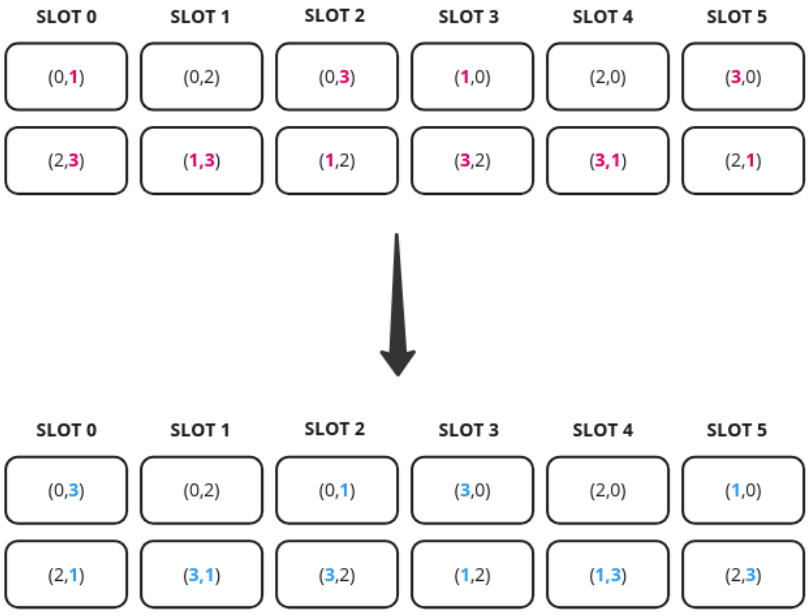
- $X'_s = \{(i,j,s')|(i,j,s) \in X_s\}$ είναι τα παιχνίδια της χρονοθυρίδας s που έγιναν παιχνίδια της χρονοθυρίδας s'
- $X'_{s'} = \{(i,j,s)|(i,j,s') \in X_s\}$ είναι τα παιχνίδια της χρονοθυρίδας s' που έγιναν παιχνίδια της χρονοθυρίδας s.

Στο Σχήμα 11 δίνεται η σχηματική αναπαράσταση της εφαρμογής του τελεστή *swap_rounds* στις χρονοθυρίδες 1, 3 ενός ωρολογίου προγράμματος αναφοράς. Αν το ωρολόγιο πρόγραμμα X είναι σε φάσεις, τότε η επιλογή των χρονοθυρίδων s, s' γίνεται από την ίδια φάση έτσι, ώστε να μην παραβιάζεται ο κανόνας των φάσεων.

Το μήκος της γειτονιάς του X που παράγεται με αυτόν τον τελεστή ισούται με

$$|\Re_x| = \begin{cases} 2 \cdot \binom{P/2}{2}, & \text{in phased timetables} \\ \binom{P}{2}, & \text{in non-phased timetables} \end{cases} \tag{59}$$

όπου P είναι το πλήθος των χρονοθυρίδων.



Σχήμα 12: Εφαρμογή του τελεστή swap_teams στις ομάδες 1, 3.

5.6.3 Τελεστής swap_teams

Ο ευρετικός τελεστής *swap_teams* s_t αλλάζει θέση σε δύο ομάδες $i, j \in T, i \neq j$ σε όλα τα παιχνίδια ενός ωρολογίου προγράμματος X. Ορίζεται ως εξής:

$$s_t \langle X|i,j\rangle = \{(x',y',s)|(x,y,s) \in X, \forall s \in S\} \tag{60}$$

όπου

$$\text{x'} = \begin{cases} j, & \text{if } x = i \\ i, & \text{if } x = j \\ x, & \text{otherwise} \end{cases} \qquad \text{y'} = \begin{cases} j, & \text{if } x = i \\ i, & \text{if } x = j \\ y, & \text{otherwise} \end{cases}$$

Στο Σχήμα 12 φαίνεται η σχηματική αναπαράσταση της εφαρμογής του τελεστή *swap_teams* στις ομάδες 1, 3 ενός ωρολογίου προγράμματος αναφοράς. Ο τελεστής *swap_teams* έχει το ίδιο μήκος γειτονιάς με τον τελεστή *swap_homes*.

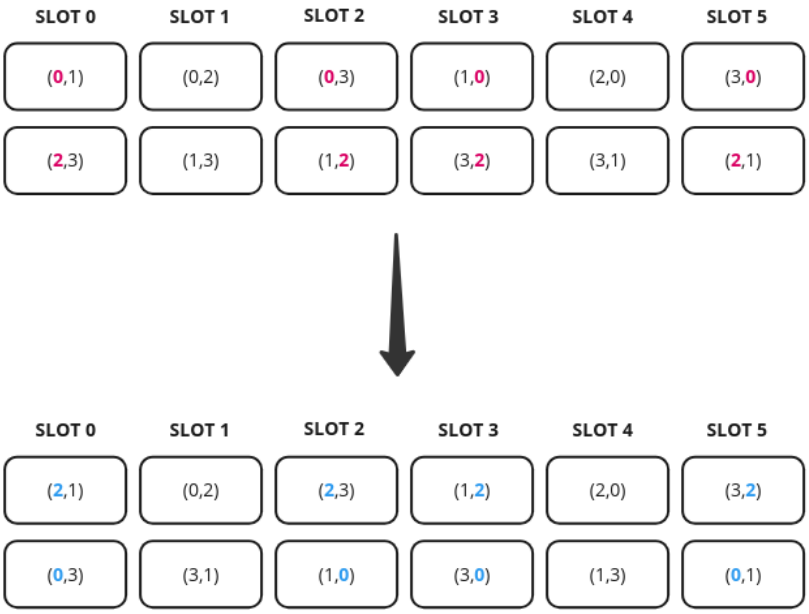
5.6.4 Τελεστής swap_opponents

Ο ευρετικός τελεστής *swap_opponents* s_p αλλάζει την θέση των αντιπάλων δύο ομάδων $i, j \in T, i \neq j$ σε κάθε χρονοθυρίδα του ωρολογίου προγράμματος X. Ο τελεστής ορίζεται ως εξής:

$$s_p \langle X|i,j\rangle = \{(x',y',s)|(x,y,s) \in X, \forall s \in S\} \tag{61}$$

όπου

- αν $x = i$ και $y \neq j$, τότε $(x', y') = (i, l)$, όπου $(j, l) \in G$
- αν $x = j$ και $y \neq i$, τότε $(x', y') = (j, k)$, όπου $(i, k) \in G$
- αν $y = i$ και $x \neq j$, τότε $(x', y') = (l, i)$, όπου $(l, j) \in G$
- αν $y = j$ και $x \neq i$, τότε $(x', y') = (k, j)$, όπου $(k, i) \in G$
- αν το παιχνίδι (x', y') είναι ήδη ορισμένο στην χρονοθυρίδα s, τότε ορίζουμε το παιχνίδι (y', x') αντ' αυτού.



Σχήμα 13: Εφαρμογή του τελεστή swap_opponents στις ομάδες 1, 3.

και G είναι το σύνολο των παιχνιδιών. Ο τελεστής *swap_opponents* έχει το ίδιο μήκος γειτονιάς με τον τελεστή *swap_homes*. Στο Σχήμα 13 φαίνεται η σχηματική αναπαράσταση της εφαρμογής του τελεστή *swap_opponents* στις ομάδες 1, 3 ενός ωρολογίου προγράμματος αναφοράς.

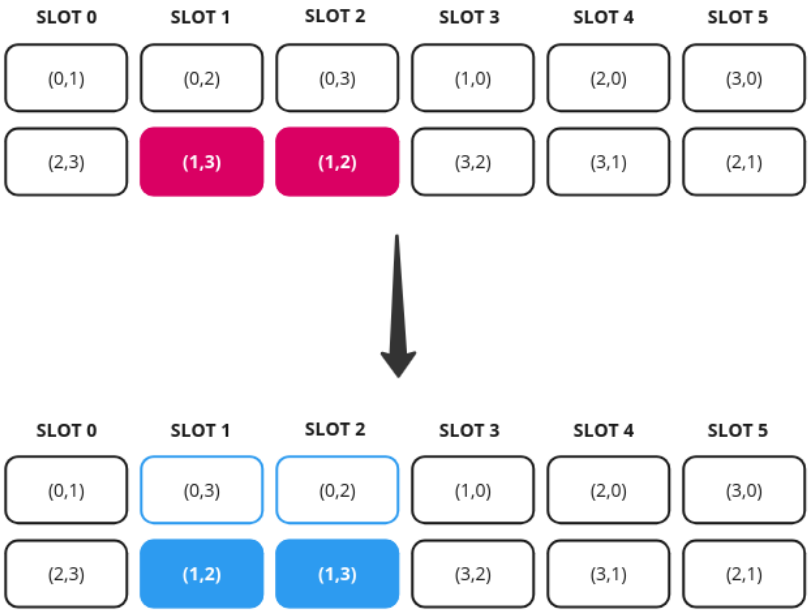
5.6.5 Τελεστής swap_matches

Ο ευρετικός τελεστής *swap_matches* s_m αλλάζει θέση σε ένα παιχνίδι (i, j) της χρονοθυρίδας $s_1 \in S$ με το αντίστοιχο στην ίδια θέση παιχνίδι της χρονοθυρίδας $s_2 \in S$. Έπειτα ελέγχεται το ωρολόγιο πρόγραμμα και αν δεν ικανοποιείται ο κανόνας 0, γίνονται τόσες ανταλλαγές στα υπόλοιπα παιχνίδια των χρονοθυρίδων s_1, s_2 έτσι, ώστε να επανέλθει η κανονικότητα. Ο τελεστής ορίζεται ως εξής:

$$s_m \langle X | i, j, s_1, s_2 \rangle = (X - (Y_s \cup Y_{s'})) \cup (Y'_s \cup Y'_{s'}) \tag{62}$$

όπου

- $Y_s \subseteq X_s$ που περιλαμβάνει το παιχνίδι (i, j) και όλα εκείνα τα παιχνίδια της χρονοθυρίδας s που μεταφέρονται στην χρονοθυρίδα s' έτσι, ώστε να τηρείται ο κανόνας 0.



Σχήμα 14: Εφαρμογή του τελεστή swap_matches στο παιχνίδι (1,3) της χρονοθυρίδας 1 με την χρονοθυρίδα 2.

- $Y_{s'} \subseteq X_{s'}$ που περιλαμβάνει όλα εκείνα τα παιχνίδια της χρονοθυρίδας s' που μεταφέρονται στην χρονοθυρίδα s έτσι, ώστε να τηρείται ο κανόνας 0, και
- $Y'_s, Y'_{s'}$ είναι τα νέα παιχνίδια της χρονοθυρίδας s και s' αντίστοιχα.

Αν το ωρολόγιο πρόγραμμα X είναι σε φάσεις, τότε η επιλογή των χρονοθυρίδων s_1, s_2 γίνεται από την ίδια φάση έτσι, ώστε να μην παραβιάζεται ο κανόνας των φάσεων.

Σε ωρολόγια προγράμματα μικρής έκτασης, ο τελεστής *swap_matches* καταλήγει να ισοδυναμεί με τον τελεστή *swap_rounds*. Καθώς το πλήθος των ανταλλαγών που γίνονται, προκειμένου να ικανοποιείται ο κανόνας 0 δεν είναι δεδομένος, το μήκος γειτονιάς του συγκεκριμένου τελεστή είναι $|\Re_x| \geq |\Re_x|_{swaprounds}$.

Στο Σχήμα 14 φαίνεται η σχηματική αναπαράσταση της εφαρμογής του ευρετικού τελεστή *swap_matches* στο παιχνίδι (1,3) της χρονοθυρίδας 1 με την χρονοθυρίδα 2 ενός ωρολογίου προγράμματος αναφοράς. Να παρατηρήσουμε ότι εκτός από την ανταλλαγή των παιχνιδιών (1,3) και (1,2) άλλαξαν θέση και τα υπόλοιπα παιχνίδια των χρονοθυρίδων έτσι, ώστε να παραμένει το νέο σωματίδιο κανονικό.

5.6.6 Τελεστής rotate_teams

Ο τελεστής *rotare_teams* r_t αλλάζει θέση με κυκλικό τρόπο σε όλες τις ομάδες του ωρολογίου προγράμματος. Ο τελεστής ορίζεται ως εξής:

$$r_t \langle X|step\rangle = \{(i', j', s)|(i, j, s) \in G, \forall i, j \in T, i \neq j\} \tag{63}$$

όπου

- $step$ είναι το βήμα κυκλικής μετατόπισης. Αν είναι θετικό, τότε η κυκλική μετατόπιση γίνεται προς τα εμπρός, ενώ αν είναι αρνητικό, προς τα πίσω.
- $i' = (i + step)mod(N)$,
- $j' = (j + step)mod(N)$

Το μήκος γειτονιάς του τελεστή *rotate_teams* ισούται με $N - 1$. Στο Σχήμα 15 φαίνεται η σχηματική αναπαράσταση της εφαρμογής του τελεστή για $step = +1$ σε ένα ωρολόγιο πρόγραμμα αναφοράς.

Σχήμα 15: Εφαρμογή του τελεστή rotate_teams για βήμα ίσο με +1.

5.6.7 Ιδιότητες τελεστών

Οι ευρετικοί τελεστές έχουν τις ακόλουθες ιδιότητες:

1. **Αντιμεταθετική ιδιότητα**: οι τελεστές *swap_homes, swap_rounds, swap_teams, swap_opponents* ικανοποιούν την αντιμεταθετική ιδιότητα ως εξής:

$$s_h \langle X|i,j\rangle = s_h \langle X|j,i\rangle$$

$$s_r \langle X|s,s'\rangle = s_r \langle X|s',s\rangle$$

$$s_t \langle X|i,j\rangle = s_t \langle X|j,i\rangle$$

$$s_p \langle X|i,j\rangle = s_p \langle X|j,i\rangle$$

2. **Ενείλιση** (*involution*): οι τελεστές *swap_homes, swap_rounds, swap_teams, swap_opponents* είναι **ενειλικτικοί**, δηλαδή ισχύει ότι:

$$s_h \langle s_h \langle X|i,j\rangle\rangle = X$$

$$s_r \langle s_r \langle X|s,s'\rangle\rangle = X$$

$$s_t \langle s_t \langle X|i,j\rangle\rangle = X$$

$$s_p \langle s_p \langle X|i,j\rangle\rangle = X$$

   Εξαιτίας αυτής της ιδιότητας, οι ενειλικτικοί τελεστές δεν επαρκούν για να διασχίσουν πλήρως τον χώρο αναζήτησης. Στην εργασία τους οι Costa κ.ά. (2012) απέδειξαν αυτή την πρόταση για τους ευρετικούς τελεστές *swap_homes*, *swap_rounds* και *swap_teams*.

3. **Αντίστροφος τελεστής**: Ο τελεστής *rotare_teams* έχει αντίστροφο τελεστή, ο οποίος είναι ο τελεστής με το αντίθετο βήμα. Δηλαδή:

$$r_t \langle r_t \langle X|step\rangle | -step\rangle = X$$

4. Όλοι οι ευρετικοί τελεστές, αν εφαρμοστούν σε κανονικά σωματίδια, παράγουν κανονικά σωματίδια.

5. **Απόσταση μεταλλαγμένου σωματιδίου**: Πειραματικές δοκιμές έδειξαν ότι οι ευρετικοί τελεστές *swap_homes, swap_matches, swap_rounds,* όταν εφαρμοστούν σε κανονικά σωματίδια, παράγουν μεταλλάξεις στην εγγύτερη γειτονιά του σωματιδίου. Αντιθέτως, οι τελεστές *swap_opponents* και *swap_teams* παράγουν μεταλλάξεις στην ευρύτερη γειτονιά του κανονικού σωματιδίου, ενώ ο τελεστής *rotate_teams* πα-

ράγει μεταλλαγμένα σωματίδια που απέχουν την μεγαλύτερη απόσταση από το αρχικό, κανονικό σωματίδιο (Πίνακας 3).

6. **Βάρος ευρετικού τελεστή**: Έστω $P_{success}$ η πιθανότητα ένας ευρετικός τελεστής που εφαρμόζεται σε ένα κανονικό σωματίδιο να παράξει ένα καλύτερο κανονικό σωματίδιο. Η πιθανότητα αυτή μπορεί να μετρηθεί πειραματικά και, όταν κανονικοποιηθεί, μας δίνει το **βάρος** του ευρετικού τελεστή. Το άθροισμα των βαρών όλων των ευρετικών τελεστών ισούται με την μονάδα.

   Το βάρος του τελεστή είναι ένα χρήσιμο μέγεθος που θα αξιοποιηθεί για να ορίσει την πιθανότητα με την οποία θα γίνεται η επιλογή ενός ευρετικού τελεστή κατά την διάρκεια τυχαίων μεταλλάξεων. Το βάρος κάθε τελεστή φαίνεται στην τελευταία στήλη του Πίνακα 3.

Τύπος σωματιδίου	Ευρετικός τελεστής	Τύπος τελεστή	Απόσταση μεταλλαγμένου σωματιδίου	Πιθανότητα επιτυχούς μετάλλαξης (%)	Βάρος τελεστή
κανονικό	rotateTeams	αντιστρέψιμος	0.9875	37.89 ± 28.98	0.12
κανονικό	swapTeams	ενειλικτικός	0.2417	45.13 ± 15.51	0.15
κανονικό	swapOpponents	ενειλικτικός	0.2333	72.73 ± 11.92	0.23
κανονικό	swapRounds	ενειλικτικός	0.0667	44.96 ± 8.35	0.14
κανονικό	swapMatches	—	0.0667	48.62 ± 7.94	0.16
κανονικό	swapHomes	ενειλικτικός	0.0083	61.46 ± 8.33	0.20

Πίνακας 3: Μέση απόσταση μεταλλαγμένου από αρχικό σωματίδιο και βάρος ευρετικού τελεστή

5.7 Μετάλλαξη και διασταύρωση

Οι ευρετικοί τελεστές που εφαρμόζονται σε ένα σωματίδιο παράγουν μεταλλαγμένα σωματίδια στην (εγγύτερη ή ευρύτερη) γειτονιά του σωματιδίου. Με τον τρόπο αυτό επιτυγχάνεται μια μεταβολή του σωματιδίου που ισοδυναμεί με μετατόπιση στον χώρο αναζήτησης και παίζει τον ρόλο της ταχύτητας στο δεξί μέλος της σχέσης (3).

Η επίδραση, όμως της μνήμης *pBest* του σωματιδίου στην ταχύτητά του και επιπροσθέτως της μνήμης *gBest* του σμήνους πώς αντιμετωπίζεται; Για τον σκοπό αυτό, ορίζουμε έναν νέο ευρετικό τελεστή: τον **τελεστή διασταύρωσης**.

5.7.1 Τελεστής crossover_match

Ο ευρετικός τελεστής διασταύρωσης, *crossover_match* c_m μεταφέρει πληροφορία από ένα σωματίδιο Y σε ένα σωματίδιο X ως εξής:

$$c_m \langle X|Y(i,j,s)\rangle = s_r \langle X|s,s'\rangle \tag{64}$$

όπου

- (i,j) είναι ένα παιχνίδι του ωρολογίου προγράμματος Y που παίζεται στην χρονοθυρίδα s.
- s' είναι η χρονοθυρίδα του ωρολογίου προγράμματος X στην οποία παίζεται το παιχνίδι (i,j).

Αυτό που μας λέει η σχέση (64) είναι ότι η πληροφορία του ωρολογίου προγράμματος Y πως το παιχνίδι (i,j) παίζεται στην χρονοθυρίδα s «*μεταφέρεται*» στο ωρολόγιο πρόγραμμα X με την βοήθεια του τελεστή *swap_rounds,* ο οποίος αλλάζει θέση σε όλα τα παιχνίδια των χρονοθυρίδων s και s', όπου s' είναι η χρονοθυρίδα στην οποία παίζεται το παιχνίδι (i,j) στο X. Με αυτό τον τρόπο, το ωρολόγιο πρόγραμμα X καταλήγει να έχει το παιχνίδι (i,j) στην χρονοθυρίδα s.

Αν το ωρολόγιο πρόγραμμα X είναι σε φάσεις, αλλά οι χρονοθυρίδες $s \in X$ και $s' \in Y$ δεν ανήκουν στην ίδια φάση, τότε εκτελείται αλλαγή φάσεων (*swap phases*) στο X, και μετά γίνεται η ανταλλαγή των αντίστοιχων χρονοθυρίδων με τον τελεστή *swap_rounds.* Αυτό γίνεται προκειμένου το νέο σωματίδιο που θα δημιουργηθεί να εξακολουθεί να είναι κανονικό.

Με την βοήθεια του ευρετικού τελεστή *crossover_match* μπορούμε να μεταφέρουμε πληροφορία σε κάθε σωματίδιο του σμήνους από την μνήμη του σωματιδίου (*pBest*) ή από την μνήμη του σμήνους (*gBest*) σε κάθε γενιά. Οι μηχανισμοί αυτοί παίζουν τον ρόλο του γνωστικού και του κοινωνικού όρου στην εξίσωση της μεταβολής της ταχύτητας του σωματιδίου (σχέση 3).

5.8 Πλήρεις τελεστές

Οι πλήρεις τελεστές είναι ευρετικοί τελεστές στους οποίους δοκιμάζονται όλοι οι πιθανοί συνδυασμοί τιμών των ορισμάτων των απλών ευρετικών τελεστών και επιστρέφεται το καλύτερο σωματίδιο που έχει βρεθεί. Πιο αναλυτικά, οι πλήρεις τελεστές είναι οι εξής:

5.8.1 Πλήρης τελεστής swap_homes_complete

Ο πλήρης ευρετικός τελεστής *swap_homes_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$s_h^c \langle X \rangle = \min \{ s_h \langle X | i, j \rangle \ \forall i, j \in T, i < j \} \tag{65}$$

όπου T είναι το σύνολο των ομάδων. Έτσι, για κάθε συνδυασμό ομάδων $i, j \in T,\ i < j$ εκτελείται ο τελεστής *swap_homes* στο X και προκύπτει ένα νέο ωρολόγιο πρόγραμμα X'. Από το σύνολο των X' επιστρέφουμε το καλύτερο ωρολόγιο πρόγραμμα σύμφωνα με το μέτρο σύγκρισης που έχουμε ορίσει στο κεφάλαιο 5.4.

5.8.2 Πλήρης τελεστής swap_rounds_complete

Ο πλήρης ευρετικός τελεστής *swap_rounds_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$s_r^c \langle X \rangle = \min \{ s_r \langle X | s_1, s_2 \rangle, \forall s_1, s_2 \in S,\ s_1 < s_2 \} \tag{66}$$

Αν το ωρολόγιο πρόγραμμα X είναι σε φάσεις, τότε ο παραπάνω ορισμός γίνεται:

$$\mathrm{s}_r^c \langle X \rangle = \min \left\{ s_r \langle X \mid s_1, s_2 \rangle, \begin{array}{l} \forall s_1, s_2 \in S,\ s_1 < s_2 : \\ s_1, s_2 \text{ in same phase} \end{array} \right\} \tag{67}$$

όπου S είναι το σύνολο των χρονοθυρίδων.

5.8.3 Πλήρης τελεστής swap_teams_complete

Ο πλήρης ευρετικός τελεστής *swap_teams_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$s_t^c \langle X \rangle = \min \{ s_t \langle X | i, j \rangle, \forall i, j \in T,\ i < j \} \tag{68}$$

όπου T είναι το σύνολο των ομάδων.

5.8.4 Πλήρης τελεστής swap_opponents_complete

Ο πλήρης ευρετικός τελεστής *swap_opponents_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$s_p^c \langle X \rangle = \min \{ s_p \langle X | i, j \rangle, \ , j \in T,\ i < j \} \tag{69}$$

όπου T είναι το σύνολο των ομάδων.

5.8.5 Πλήρης τελεστής swap_matches_complete

Ο πλήρης ευρετικός τελεστής *swap_matches_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$\mathrm{s}_m^c \langle X \rangle = \min \left\{ s_m \langle X \mid i, j, s_1, s_2 \rangle, \begin{array}{l} \forall (i,j) \in G_{s1}, \\ \forall s_1, s_2 \in S, \\ s_1 < s_2 \end{array} \right\} \tag{70}$$

όπου G_{s1} είναι το σύνολο των αγώνων του ωρολογίου προγράμματος X που λαμβάνουν χώρα στην χρονοθυρίδα s_1 και S είναι το σύνολο των χρονοθυρίδων. Για την ειδική περίπτωση που το ωρολόγιο πρόγραμμα X είναι σε φάσεις, ο ορισμός γίνεται ως εξής:

$$\mathrm{s}_m^c \langle X \rangle = \min \left\{ s_m \langle X \mid i, j, s_1, s_2 \rangle, \ \forall (i,j) \in G_{s1}, \quad \begin{array}{l} \forall s_1, s_2 \in S, \\ s_1 < s_2, \\ s_1, s_2 \text{ in same phase} \end{array} \right\} \tag{71}$$

5.8.6 Πλήρης τελεστής rotate_teams_complete

Ο πλήρης ευρετικός τελεστής *rotate_teams_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$\mathrm{r}_t^c \langle X \rangle = \min \{ r_t \langle X \mid step \rangle, \ r_t \langle X \mid -step \rangle, \ \forall step \in [1, |T| - 1] \} \tag{72}$$

όπου η μεταβλητή *step* είναι θετικός, ακέραιος αριθμός, και $|T|$ είναι το πλήθος των ομάδων.

5.8.7 Πλήρης τελεστής crossover_match_complete

Ο πλήρης ευρετικός τελεστής *crossover_match_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p ορίζεται ως εξής:

$$c_m^c \langle X | Y \rangle = \min \{ c_m \langle X | Y(i,j,s) \rangle, \ \forall (i,j) \in G_s, \ \forall s \in S \} \tag{73}$$

όπου G_s είναι το σύνολο των αγώνων του ωρολογίου προγράμματος X που λαμβάνουν χώρα στην χρονοθυρίδα s και S είναι το σύνολο των χρονοθυρίδων.

5.8.8 Πλήρης τελεστής swap_matches_crossover_complete

Ο πλήρης ευρετικός τελεστής *swap_matches_crossover_complete* του ωρολογίου προγράμματος X ενός σωματιδίου p συνδυάζει τον πλήρη τελεστή *swap_matches_complete* με τον πλήρη τελεστή *crossover_match_complete* και ορίζεται ως εξής:

$$\mathrm{s}^{c}_{m,cr}\left\langle X\right\rangle = \min\left\{ c^{c}_{m}\left\langle s_m\left\langle X \mid i,j,s_1,s_2\right\rangle \mid X\right\rangle, \begin{array}{l} \forall (i,j)\in G_{s1}, \\ \forall s_1,s_2\in S, \\ s_1<s_2 \end{array} \right\} \tag{74}$$

όπου G_{s1} είναι το σύνολο των αγώνων του ωρολογίου προγράμματος X που λαμβάνουν χώρα στην χρονοθυρίδα s_1 και S είναι το σύνολο των χρονοθυρίδων.

Δηλαδή για κάθε μεμονωμένο *swap_matches* που εκτελείται παράγεται ένα ενδιάμεσο ωρολόγιο πρόγραμμα, το οποίο στην συνέχεια γίνεται *crossover_match_complete* με το αρχικό ωρολόγιο πρόγραμμα X. Από το σύνολο των νέων, ωρολογίων προγραμμάτων που προκύπτουν με αυτό τον τρόπο, επιστρέφεται το καλύτερο.

5.9 Εξελιγμένο μοντέλο PSO

Χρησιμοποιώντας τα εργαλεία που ορίσαμε παραπάνω, μπορούμε να εξελίξουμε το αρχικό μοντέλο PSO στον διακριτό χώρο και να ορίσουμε την μετατόπιση $\Delta\overrightarrow{x_i}$ του σωματιδίου i του σμήνους στον χώρο αναζήτησης ως εξής:

$$\Delta\overrightarrow{x_i} = \underbrace{P_w\overrightarrow{v_i}}_{\text{mutation}} + \underbrace{P_c(\overrightarrow{p_i}-\overrightarrow{x_i})}_{\text{crossover with pBest}} + \underbrace{P_s(\overrightarrow{g_i}-\overrightarrow{x_i})}_{\text{crossover with gBest}} \tag{75}$$

όπου

- $\overrightarrow{v_i}$ είναι το διάνυσμα της ταχύτητας του σωματιδίου.
- $\overrightarrow{p_i}$ είναι το διάνυσμα της θέσης του σωματιδίου *pBest*.
- $\overrightarrow{g_i}$ είναι το διάνυσμα της θέσης του σωματιδίου *gBest*.
- P_w είναι η πιθανότητα μετάλλαξης του σωματιδίου.
- P_c είναι η πιθανότητα διασταύρωσης του σωματιδίου με το *pBest*.

- P_s είναι η πιθανότητα διασταύρωσης του σωματιδίου με το *gBest*.

Ο πρώτος όρος στο δεξί μέλος της σχέσης (75) αντιπροσωπεύει την μετάλλαξη του σωματιδίου, ο δεύτερος όρος την διασταύρωση του σωματιδίου με το σωματίδιο *pBest*, και ο τρίτος όρος την διασταύρωση του σωματιδίου με το σωματίδιο *gBest*.

Εύκολα μπορούμε να διαπιστώσουμε από την σχέση (2) ότι η μετατόπιση $\Delta\overrightarrow{x_i}$ ισοδυναμεί με την ταχύτητα $v_i(t+1)$ του σωματιδίου την χρονική στιγμή $t+1$. Έτσι, με την βοήθεια της σχέσης (6) προκύπτει ότι:

$$P_w \to w$$
$$P_c \to \varphi_p$$
$$P_s \to \varphi_g$$

όπου οι τυχαίες μεταβλητές r_p, r_g που είχαν εισαχθεί στο κανονικό μοντέλο PSO για να προσφέρουν τυχαιότητα έχουν κανονικοποιηθεί στην μονάδα.

Συνάμα, από τον ορισμό της πιθανότητας ισχύει ότι:

$$P_w + P_c + P_s = 1$$

Με αυτά κατά νου, και χρησιμοποιώντας τους συντελεστές συμπίεσης (σχέσεις 8 & 9) προκύπτει ότι:

$$P_c = P_s = \varphi P_w$$

και επομένως,

$$P_w = \frac{1}{1+2\varphi} \tag{76}$$

$$P_c = P_s = \frac{\varphi}{1+2\varphi} \tag{77}$$

Έτσι, στο νέο εξελιγμένο μοντέλο PSO (*Advanced PSO*), η μετατόπιση ενός σωματιδίου κάθε χρονική στιγμή στο σμήνος καθορίζεται πιθανολογικά από την μετάλλαξή του, την

διασταύρωσή του με την καλύτερη μέχρι στιγμής εκδοχή του, και την διασταύρωσή του με την καλύτερη μέχρι στιγμής εκδοχή του σμήνους στο οποίο ανήκει.

Στον Πίνακα 4 δίνεται η ανάλυση των τιμών των πιθανοτήτων μετάλλαξης και διασταύρωσης ενός σωματιδίου ως συνάρτηση του συντελεστή συμπίεσης φ. Να υπενθυμίσουμε ότι σύμφωνα με την σχέση (8) ο συντελεστής συμπίεσης φ πρέπει να είναι πάντοτε μεγαλύτερος του 2.

φ	w	φ_p	φ_g	P_w	P_c	P_s
2,00	1,000	2,000	2,000	0,2000	0,4000	0,4000
2,01	0,868	1,745	1,745	0,1992	0,4004	0,4004
2,02	0,819	1,654	1,654	0,1984	0,4008	0,4008
2,03	0,783	1,590	1,590	0,1976	0,4012	0,4012
2,04	**0,754**	**1,539**	**1,539**	**0,1969**	**0,4015**	**0,4015**
2,05	0,730	1,496	1,496	0,1961	0,4020	0,4020
2,06	0,708	1,459	1,459	0,1953	0,4023	0,4023
2,07	0,689	1,427	1,427	0,1946	0,4027	0,4027
2,08	0,672	1,398	1,398	0,1938	0,4031	0,4031
2,09	0,656	1,372	1,372	0,1931	0,4035	0,4035
2,10	0,642	1,348	1,348	0,1923	0,4038	0,4038
2,20	0,537	1,181	1,181	0,1852	0,4074	0,4074
2,30	0,469	1,079	1,079	0,1786	0,4107	0,4107
2,40	0,420	1,008	1,008	0,1724	0,4138	0,4138
2,50	0,382	0,955	0,955	0,1667	0,4167	0,4167
2,60	0,351	0,913	0,913	0,1613	0,4194	0,4194
2,70	0,325	0,878	0,878	0,1563	0,4219	0,4219
2,80	0,303	0,849	0,849	0,1515	0,4242	0,4242
2,90	0,284	0,825	0,825	0,1471	0,4265	0,4265

Πίνακας 4: Τιμές πιθανοτήτων μετάλλαξης και διασταύρωσης ως συνάρτηση της παραμέτρου φ. Με έντονη γραφή σημειώνονται οι τιμές των πιθανοτήτων που με πειραματικές δοκιμές επιλέχθηκαν στην υλοποίηση του αλγορίθμου CP-PSO.

□

6 Υβριδική προσέγγιση

Ο αλγόριθμος CP-PSO που προτείνουμε είναι ένας υβριδικός αλγόριθμος που συνδυάζει την τεχνολογία Constraint Programming με την δυναμική του αλγορίθμου Particle Swarm Optimization. Ο αλγόριθμος επιμερίζει το πρόβλημα χρονοπρογραμματισμού αθλητικών αγώνων, όπως αυτά του οργανισμού ITC2021, σε δύο φάσεις (Σχήμα 16):

1. Την φάση ικανοποιησιμότητας (*Satisfiability Phase*), και
2. Την φάση βελτιστοποίησης (*Optimization Phase*)

Σε ό,τι ακολουθεί θα μελετήσουμε ξεχωριστά αυτές τις φάσεις.

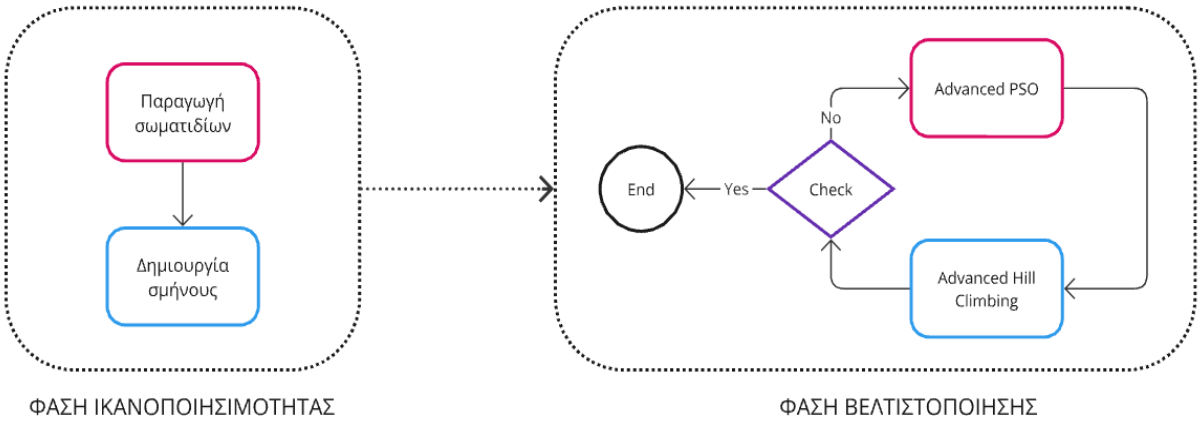
6.1 Φάση Ικανοποιησιμότητας

Η φάση της ικανοποιησιμότητας είναι η φάση στην οποία παράγονται τα αρχικά σωματίδια του σμήνους. Ανάλογα με την πολυπλοκότητα του προβλήματος, στην φάση αυτή επιλέγουμε τους περιορισμούς που επιθυμούμε να ικανοποιούν πλήρως τα αρχικά σωματίδια. Υπάρχουν δύο επιλογές:

- Τα αρχικά σωματίδια του σμήνους να είναι εφικτά. Για την παραγωγή τους πρέπει να ικανοποιούνται οι δομικοί και ισχυροί περιορισμοί του προβλήματος.
- Τα αρχικά σωματίδια του σμήνους να είναι κανονικά, αλλά όχι εφικτά. Στην περίπτωση αυτή θα πρέπει να ικανοποιούνται μόνον οι δομικοί περιορισμοί του προβλήματος.

Για την παραγωγή των σωματιδίων, επιστρατεύουμε τον αλγόριθμο ανοικτού κώδικα της Google, CP-SAT. Το πλεονέκτημα του CP-SAT έναντι άλλων CP ευρετικών είναι πως διαμορφώνει το πρόβλημα χρονοπρογραμματισμού αθλητικών αγώνων σε ένα μοντέλο ικανοποιησιμότητας ανάγοντας τους ισχυρούς περιορισμούς σε λογικές προτάσεις, τακτική που φαίνεται να δίνει καλύτερα αποτελέσματα (Dimitsas κ.ά., 2022).

Αναπτύξαμε τον CP-SAT Generator ως μια γεννήτρια παραγωγής σωματιδίων. Ο CP-SAT Generator λαμβάνει τα αρχικά δεδομένα του προβλήματος και έχει την δυνατότητα να παράξει κανονικά ή εφικτά σωματίδια με άμεση εφαρμογή των περιορισμών που απαιτούνται. Αναλυτικά, ο κώδικας της γεννήτριας περιγράφεται στο κεφάλαιο 7.3.



Σχήμα 16: Υβριδική προσέγγιση δύο φάσεων

6.1.1 Παραγωγή κανονικών σωματιδίων

Τα κανονικά σωματίδια παράγονται με εισαγωγή μόνον των δομικών περιορισμών στο μοντέλο του CP-SAT. Η γεννήτρια σωματιδίων CP-SAT Generator (κεφάλαιο 7.3) επιλύει αυτό το πρόβλημα σε συνήθως σύντομο χρονικό διάστημα.

6.1.2 Παραγωγή εφικτών σωματιδίων

Τα εφικτά σωματίδια απαιτούν την ικανοποίηση των δομικών και των ισχυρών περιορισμών του προβλήματος. Για την παραγωγή ενός εφικτού σωματιδίου, ακολουθούμε δύο βήματα:

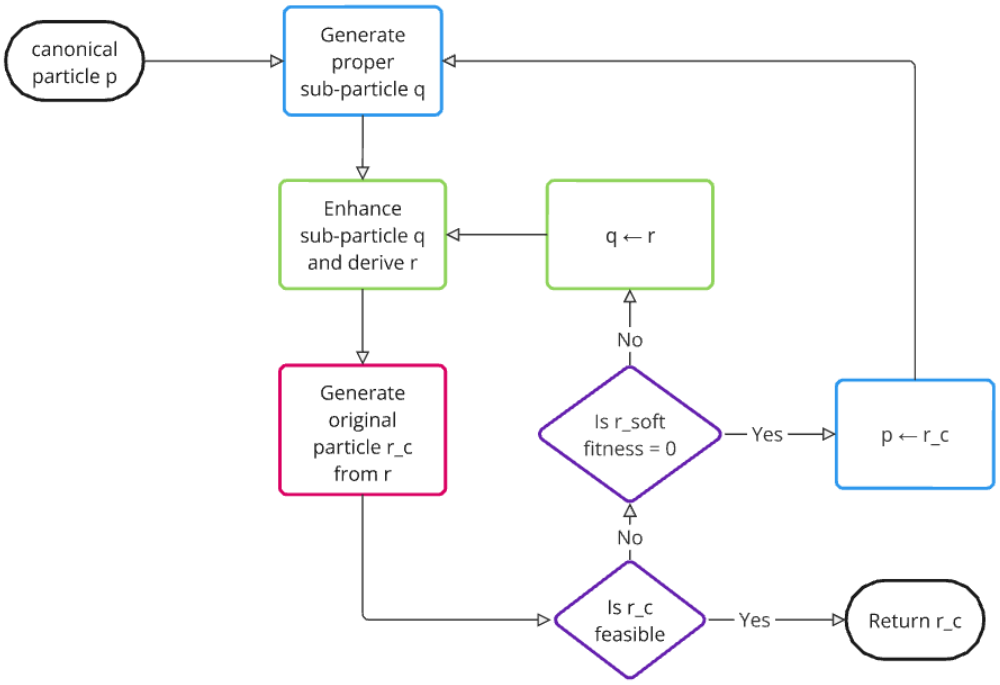
1. Παράγουμε ένα κανονικό σωματίδιο χρησιμοποιώντας τον CP-SAT Generator.
2. Βελτιώνουμε το κανονικό σωματίδιο με την βοήθεια του αλγόριθμου βελτιστοποίησης CP-SAT Enhancer, τον οποίον θα δούμε αναλυτικά στο κεφάλαιο 6.2.2.

Το πλεονέκτημα αυτής της μεθόδου είναι πως συνήθως παράγεται πολύ γρήγορα ένα κανονικό σωματίδιο, το οποίο στην συνέχεια βελτιώνεται σταδιακά χρησιμοποιώντας την ευρετική *fix-and-optimize* (την γενική έννοια της οποίας ορίσαμε στο κεφάλαιο 3.3). Η ευρετική αυτή έχει την δυνατότητα να εξερευνά τον χώρο αναζήτησης με μια ακολουθία απλοποιημένων προβλημάτων που προέρχονται από το αρχικό μας πρόβλημα με την μετατροπή μέρους των μεταβλητών απόφασης σε σταθερές.

Εφαρμογή γνήσιων υποσωματιδίων

Αν η παραγωγή ενός εφικτού σωματιδίου με την παραπάνω διαδικασία είναι επίπονη, τότε ενδέχεται να επιτύχουμε καλύτερα αποτελέσματα χρησιμοποιώντας γνήσια υποσωματίδια. Με την μέθοδο αυτή, από το κανονικό σωματίδιο του αρχικού προβλήματος παράγεται ένα γνήσιο υποσωματίδιο, η βελτίωση του οποίου βελτιώνει το αρχικό πρόβλημα. Η διαδικασία επαναλαμβάνεται έως ότου το κανονικό σωματίδιο του αρχικού προβλήματος γίνει εφικτό.

Στο Σχήμα 17 φαίνεται η διαδικασία παραγωγής εφικτού σωματιδίου από ένα κανονικό σωματίδιο p. Αρχικώς, παράγουμε ένα γνήσιο υποσωματίδιο q. Για την παραγωγή του, λαμβάνουμε ένα κλάσμα από τους ισχυρούς περιορισμούς που παραβιάζει το σωματίδιο p. Συνήθως, επιλέγουμε λίγους περιορισμούς (2 ή 3). Στην συνέχεια, βελτιώνουμε το γνήσιο υποσωματίδιο με την βοήθεια του CP-SAT Enhancer παράγοντας το υποσωματίδιο r.



Σχήμα 17: Παραγωγή εφικτού σωματιδίου από κανονικό με χρήση υποσωματιδίων

Με το ωρολόγιο πρόγραμμα του βελτιωμένου υποσωματιδίου r, παράγουμε ένα πρωτότυπο σωματίδιο r_c (που λαμβάνει υπ' όψιν όλους τους περιορισμούς του προβλήματος). Αν το σωματίδιο r_c είναι εφικτό, τότε έχουμε επιτύχει τον στόχο μας. Διαφορετικά, ελέγχουμε αν

το βελτιωμένο υποσωματίδιο r έχει μηδενίσει τους ασθενείς περιορισμούς του. Αν δεν το έχει κάνει, τότε συνεχίζουμε την βελτίωση πάνω στο ίδιο υποσωματίδιο. Στην αντίθετη περίπτωση, το υποσωματίδιο r έχει παράξει ένα ωρολόγιο πρόγραμμα που ικανοποιεί όλους τους ισχυρούς περιορισμούς που του δώσαμε. Έτσι, με την βοήθεια του r_c παράγουμε ένα νέο γνήσιο υποσωματίδιο και επαναλαμβάνουμε την διαδικασία.

Η βελτιστοποίηση με γνήσια υποσωματίδια είναι ένα πολύ ισχυρό εργαλείο για την παραγωγή εφικτών σωματιδίων. Η μέθοδος ελαττώνει σταδιακά το πλήθος των ισχυρών περιορισμών που παραβιάζονται οδηγώντας εν τέλει στην ικανοποίηση όλων των ισχυρών περιορισμών και στην παραγωγή ενός εφικτού σωματιδίου.

□

Από το σύνολο των σωματιδίων (κανονικών ή εφικτών) που παράγονται λαμβάνουμε ένα υποσύνολο που αποτελείται από σωματίδια, διαφορετικά μεταξύ τους, με μέση απόσταση μεγαλύτερη από 0.9. Με αυτό τον τρόπο, επιτυγχάνουμε μία ικανοποιητική διασπορά των αρχικών σωματιδίων στον χώρο αναζήτησης. Τα σωματίδια αυτά θα αποτελέσουν το αρχικό σμήνος που θα εισαχθεί στην επόμενη φάση.

Μετά την αρχικοποίηση του σμήνους, ο αλγόριθμος ενημερώνει την μνήμη του σμήνους (*gBest*) με το καλύτερο σωματίδιο που έχει βρεθεί μέχρι στιγμής.

6.2 Φάση Βελτιστοποίησης

Στην φάση αυτή, ενεργοποιούνται όλοι οι περιορισμοί του προβλήματος, ισχυροί και ασθενείς. Τα σωματίδια που έχουν παραχθεί στην φάση ικανοποιησιμότητας εισάγονται ως αρχικά δεδομένα στην παρούσα φάση.

6.2.1 Advanced PSO

Ο αλγόριθμος χρησιμοποιεί το εξελιγμένο μοντέλο PSO (*Advanced PSO*) που περιγράψαμε στο κεφάλαιο 5.9, για να κινηθεί σε εφικτά (*feasible*) και ανέφικτα (*infeasible*) τμήματα του χώρου αναζήτησης με σκοπό την εύρεση μιας βέλτιστης λύσης που να ικανοποιεί πλήρως τους δομικούς και ισχυρούς περιορισμούς και να ελαχιστοποιεί το κόστος των ασθενών περιορισμών.

Σε κάθε επανάληψη $t < t_{max}$, όπου t_{max} είναι το μέγιστο πλήθος των επαναλήψεων (γενεών), κάθε σωματίδιο p του σμήνους επιλέγει με

- πιθανότητα P_w να μεταλλαχθεί (*stochastic mutation*),
- πιθανότητα P_c να διασταυρωθεί με το *pBest* του (*stochastic crossover*), και
- πιθανότητα P_s να διασταυρωθεί με το *gBest* του σμήνους (*stochastic crossover*).

όπου οι πιθανότητες αυτές υπολογίζονται από τις σχέσεις (76) και (77) και είναι συνάρτηση του φ (Πίνακας 4).

Εφόσον επιλεγεί να γίνει μετάλλαξη του σωματιδίου, εκτελείται κάθε φορά με τυχαίο τρόπο και λαμβάνοντας υπ' όψιν τα βάρη τους ένας από τους παρακάτω ευρετικούς τελεστές:

- *rotate_teams* (συμπεριλαμβανομένου και του αντιστρόφου του),
- *swap_homes,*
- *swap_rounds,*
- *swap_teams,*
- *swap_opponents* και
- *swap_matches*

και παράγεται το μεταλλαγμένο σωματίδιο. Οι διασταυρώσεις με το *pBest* ή το *gBest*, εφόσον επιλεγούν, γίνονται με την βοήθεια του ευρετικού τελεστή *crossover_match*.

Στην συνέχεια, το σωματίδιο p' που παράχθηκε από το p ελέγχεται με βάση το μέτρο σύγκρισης που ορίσαμε στο κεφάλαιο 5.4. Αν το p' είναι καλύτερο του p, τότε γίνεται αμέσως αποδεκτό. Αν είναι χειρότερο, τότε ενεργοποιείται ένα σχήμα Simulated Annealing (SA), το οποίο σκοπό έχει να βοηθήσει το σμήνος να εξερευνήσει ανεξερεύνητα τμήματα του χώρου αναζήτησης στις πρώτες γενιές και να το ελευθερώσει από τυχόν τοπικά βέλτιστα. Το σχήμα SA αποδέχεται μία «κακή» λύση, αν για έναν τυχαίο αριθμό $r \in [0, 1]$ ισχύει ότι:

$$r < e^{-kf_w/T}$$

όπου

- $T = t_{max} - t$ είναι η τρέχουσα θερμοκρασία.

- f_w είναι το κόστος του σωματιδίου, όπως έχει διαμορφωθεί με την επιβολή ειδικών βαρών στις ποινές παραβιάσεις των ισχυρών και/ή ασθενών περιορισμών.
- k είναι μια σταθερά (η τιμή της οποίας έχει προκύψει με πειραματικές δοκιμές).

Η προηγούμενη σχέση ορίζει το *κριτήριο Metropolis*. Η θερμοκρασία T ξεκινά από υψηλά επίπεδα (τουλάχιστον $t_{max} = 10000$) και ελαττώνεται με πολύ αργό ρυθμό έως μία πολύ χαμηλή τιμή. Με την βοήθεια του σχήματος SA, ο αλγόριθμος εξερευνά (*exploration*) καλύτερα τον χώρο αναζήτησης στις αρχικές γενιές (όπου η πιθανότητα αποδοχής «*κακών*» λύσεων είναι μεγάλη) αποφεύγοντας τοπικά ελάχιστα, ενώ στις κατοπινές γενιές εστιάζει στην τοπική αναζήτηση βελτιώνοντας μία βέλτιστη λύση (*exploitation*).

Στην συνέχεια, ο αλγόριθμος ενημερώνει την μνήμη (*pBest*) του σωματιδίου όπως και την μνήμη (*gBest*) του σμήνους. Οι παραπάνω διαδικασίες επαναλαμβάνονται για όλα τα σωματίδια και για όλες τις γενιές, έως ότου λήξει ο χρόνος.

6.2.2 Advanced Hill Climbing

Έχοντας ολοκληρώσει ο αλγόριθμος Advanced PSO ένα μέγιστο πλήθος γενεών (το λιγότερο 10000) εξερευνώντας τον χώρο αναζήτησης και βελτιώνοντας τα αρχικά σωματίδια του σμήνους, περνάμε στο επόμενο στάδιο, όπου εστιάζουμε αποκλειστικά στην εκμετάλλευση (*exploitation*) του χώρου αναζήτησης με έναν βελτιωμένο μηχανισμό Hill Climbing που χρησιμοποιεί δύο πολύ δυνατά εργαλεία για να βελτιστοποιήσει τα σωματίδια του σμήνους: τον αλγόριθμο CP-SAT Enhancer και τους πλήρεις τελεστές. Έτσι, σκοπός μας είναι να βελτιώσουμε τα σωματίδια στην γειτονιά τους, προτού επιστρέψουμε ξανά στο Advanced PSO για περαιτέρω εξερεύνηση (exploration) του χώρου αναζήτησης.

Hill Climbing με CP-SAT

Σε πρώτη φάση επικαλούμαστε ένα τροποποιημένο μοντέλο CP-SAT για την περαιτέρω βελτίωση της απόδοσης των σωματιδίων του σμήνους εφαρμόζοντας την ευρετική *fix-and-optimize*. Με βάση το μοντέλο αυτό, δημιουργήσαμε τον αλγόριθμο βελτιστοποίησης CP-SAT Enhancer, το διάγραμμα ροής του οποίου φαίνεται στο Σχήμα 18. Η λειτουργία του αλγορίθμου CP-SAT Enhancer είναι η εξής:

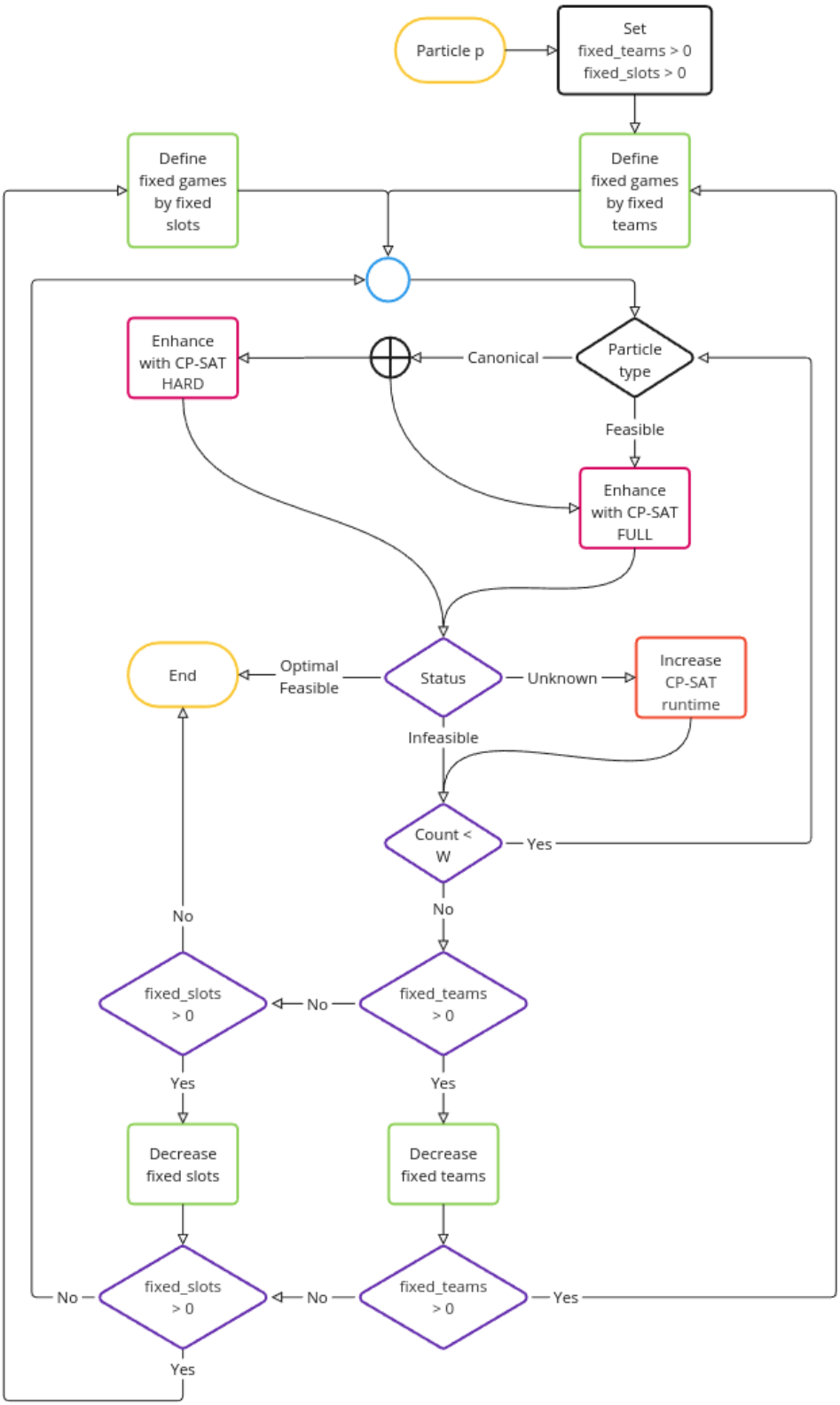
Αρχικώς, επιλέγουμε ένα μέγιστο επιτρεπτό πλήθος σταθερών ομάδων (*teams_fix_num*) και βρίσκουμε τα παιχνίδια στα οποία συμμετέχουν οι ομάδες αυτές. Τα παιχνίδια αυτά ορίζονται ως σταθερές στο μοντέλο. Στην συνέχεια,

- Αν το σωματίδιο που θέλουμε να βελτιώσουμε είναι κανονικό, αλλά μη-εφικτό, τότε έχουμε δύο επιλογές:
  - είτε εισάγουμε στο μοντέλο μόνον τους δομικούς περιορισμούς και τους ισχυρούς ως ασθενείς (*Enhance with CP-SAT Hard*), ή
  - όλους τους περιορισμούς (*Enhance with CP-SAT Full*).

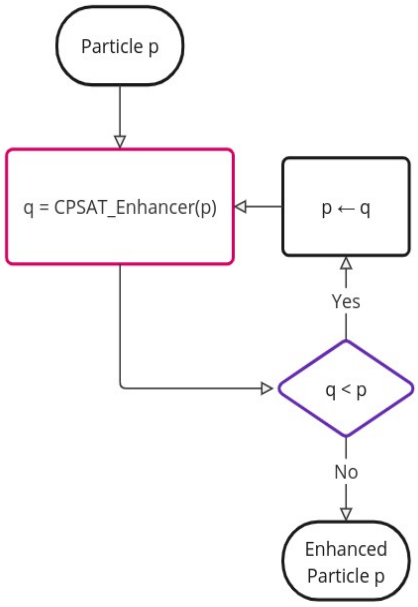
  Με την πρώτη επιλογή εστιάζουμε μόνον στην επίδραση που έχουν οι ισχυροί περιορισμοί στο συνολικό κόστος του σωματιδίου και επιχειρούμε να μηδενίσουμε το κόστος τους σταδιακά. Με την δεύτερη, το σύνολο των περιορισμών συμμετέχει στην διαδικασία βελτιστοποίησης. Ποια επιλογή θα διαλέξουμε είναι συνάρτηση της πολυπλοκότητας του προβλήματος και του υπολογιστικού κόστους.
- Αν το σωματίδιο που θέλουμε να βελτιώσουμε είναι εφικτό, τότε εισάγουμε στο μοντέλο όλους τους περιορισμούς (*Enhance with CP-SAT Full*).

Με αυτές τις παραμέτρους, αναλαμβάνει ο CP-SAT να βελτιώσει το σωματίδιο. Το *κριτήριο βελτίωσης* αφορά στην εύρεση ενός καλύτερου σωματιδίου με βάση το μέτρο σύγκρισης που έχουμε ορίσει στο κεφάλαιο 5.4, και όχι με βάση τον έμφυτο μηχανισμό βελτιστοποίησης (*minimization*) του CP-SAT, καθώς ο τελευταίος είναι εξαιρετικά χρονοβόρος σε σύνθετα προβλήματα χρονοπρογραμματισμού αγώνων και επομένως, μη αποδοτικός. Αν ο CP-SAT επιστρέψει:

1. **INFEASIBLE** που σημαίνει ότι έχει αποδείξει πως το πρόβλημα δεν έχει εφικτή λύση με τις ομάδες (*teams_fix_num*) που έχουμε ορίσει ως σταθερές, τότε και προτού τις μειώσουμε, δοκιμάζουμε ξανά για ένα συγκεκριμένο πλήθος επαναλήψεων W. Έπειτα από W αποτυχημένες επαναλήψεις, ελαττώνουμε το πλήθος των ομάδων (*teams_fix_num*) που διατηρούνται σταθερές κατά 1 και επαναλαμβάνουμε την διαδικασία.
2. **OPTIMAL/FEASIBLE**, τότε η λύση που έχει επιτευχθεί είναι βέλτιστη/εφικτή και την επιστρέφουμε στην έξοδο τερματίζοντας την διαδικασία.



Σχήμα 18: Ο αλγόριθμος βελτιστοποίησης CP-SAT Enhancer



Σχήμα 19: Hill Climbing με CP-SAT

3. **UNKNOWN**, τότε ο μέγιστος χρόνος εκτέλεσης που έχουμε ορίσει δεν είναι αρκετός για να καταλήξει ο CP-SAT σε συμπέρασμα. Να σημειώσουμε ότι η κατάσταση UNKNOWN θα επιστραφεί μόνον όταν έχει οριστεί χρονικό όριο εκτέλεσης.

Όταν το πλήθος των σταθερών ομάδων γίνει μηδέν (*teams_fix_num = 0*), ο αλγόριθμος λαμβάνει ένα μέγιστο επιτρεπτό πλήθος σταθερών χρονοθυρίδων (*slots_fix_num*) και βρίσκει τα παιχνίδια που εκτελούνται σε αυτές. Τα παιχνίδια αυτά ορίζονται τώρα ως σταθερές στο μοντέλο και η διαδικασία που περιγράψαμε παραπάνω επαναλαμβάνεται με την παράμετρο *slots_fix_num* στην θέση της *teams_fix_num.*

Όταν και το πλήθος των σταθερών χρονοθυρίδων γίνει μηδέν (*slots_fix_num = 0*), τότε δεν διατηρούμε πλέον σταθερές ούτε ομάδες, ούτε χρονοθυρίδες. Στην φάση αυτή, ο αλγόριθμος εκτελεί μία ακόμα επανάληψη αναζητώντας λύση.

Αν μετά την επανάληψη αυτή δεν βρεθεί λύση, τότε ο αλγόριθμος ολοκληρώνεται επιστρέφοντας το αρχικό σωματίδιο που εν τέλει δεν βελτιώθηκε. Αν, ωστόσο, κατά την διάρκεια εκτέλεσης της παραπάνω διαδικασίας βρεθεί λύση, τότε η διαδικασία τερματίζεται αμέσως και επιστρέφεται η λύση που βρέθηκε.

Ο αλγόριθμος βελτιστοποίησης CP-SAT Enhancer είναι μέρος ενός μηχανισμού Hill Climbing, ο οποίος λαμβάνει το αρχικό σωματίδιο και επιχειρεί να το βελτιώσει σταδιακά. Σε κάθε βήμα, ο αλγόριθμος βελτιστοποίησης CP-SAT Enhancer προσπαθεί να βελτιώσει το τρέχον σωματίδιο και αν το καταφέρει, επαναλαμβάνει την διαδικασία με όρισμα το νέο, βελτιωμένο σωματίδιο. Όταν δεν μπορεί να επιτευχθεί περαιτέρω βελτίωση, ο μηχανισμός Hill Climbing τερματίζεται και επιστρέφει το τρέχον σωματίδιο (που είναι η καλύτερη βελτίωση του αρχικού σωματιδίου που κατάφερε να επιτύχει). Σχηματικά, ο μηχανισμός Hill Climbing με CP-SAT φαίνεται στο Σχήμα 19.

Hill Climbing με πλήρεις τελεστές

Ο μηχανισμός Hill Climbing με πλήρεις τελεστές χρησιμοποιεί όλους τους πλήρεις τελεστές που ορίσαμε στο κεφάλαιο 5.8 προκειμένου να βελτιώσει τα σωματίδια του σμήνους στην γειτονιά τους. Αναλυτικά, το διάγραμμα ροής του μηχανισμού Hill Climbing με πλήρεις τελεστές φαίνεται στο Σχήμα 20.

Να τονίσουμε στο σημείο αυτό ότι η σειρά με την οποία επιλέγουμε να εφαρμόσουμε τους πλήρεις τελεστές για να βελτιώσουμε ένα σωματίδιο είναι τυχαία. Επίσης, οι πλήρεις τελεστές *swap_matches_complete* και *crossover_match_complete* συγχωνεύονται στον πλήρη τελεστή *swap_matches_crossover_complete.*

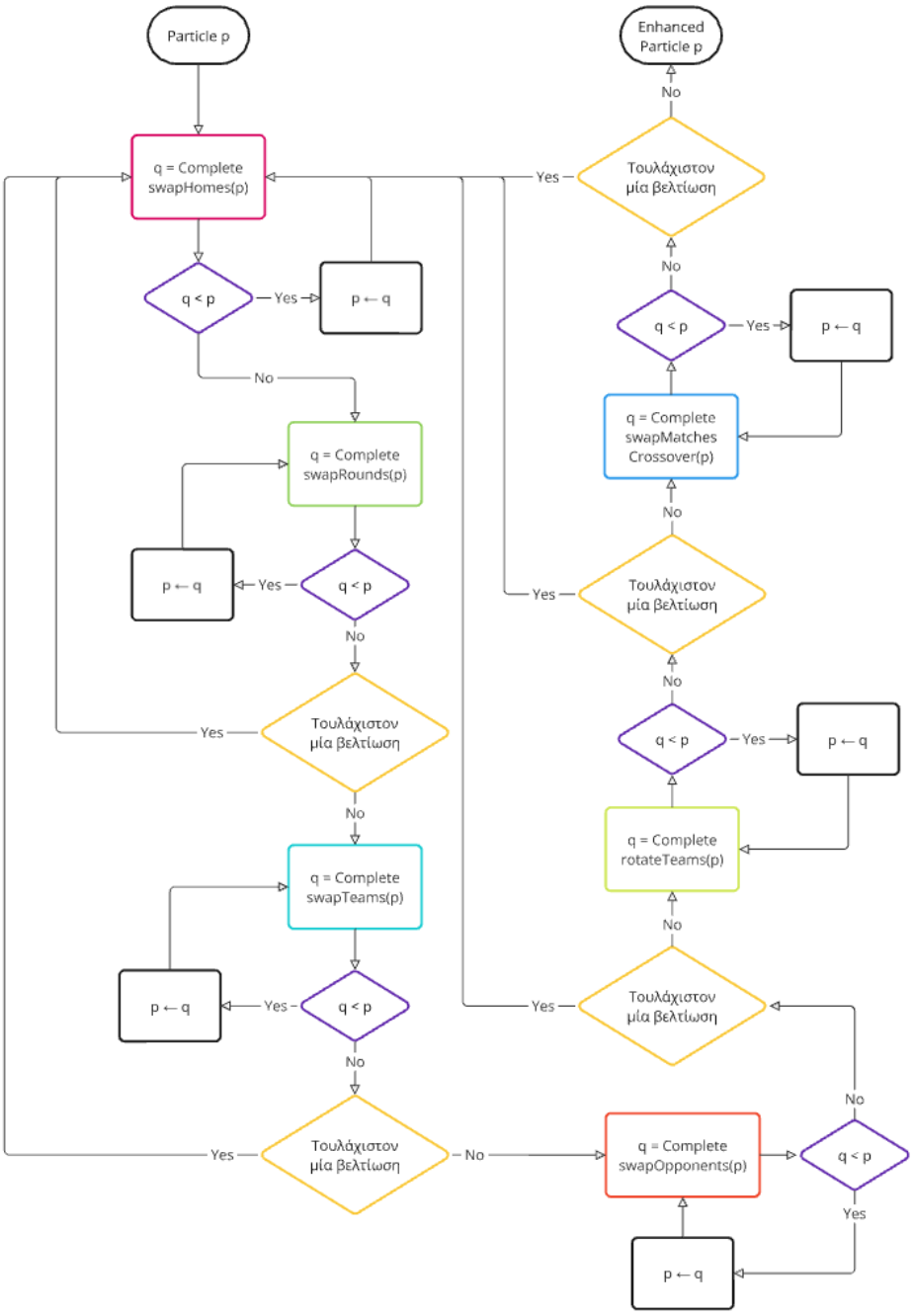
Στην επιλογή που έχουμε κάνει, κάθε φορά που υπάρχει τουλάχιστον μία βελτίωση του σωματιδίου μετά τις επανειλημμένες προσπάθειες κάθε πλήρους τελεστή (πλην του *swap_homes_complete*), ο αλγόριθμος επιστρέφει στον *swap_homes_complete* και επαναλαμβάνει την διαδικασία από την αρχή. Αυτό γίνεται διότι παρατηρήσαμε πειραματικά πως ο πλήρης τελεστής *swap_homes_complete* σε αρκετές περιπτώσεις επιτύγχανε επιπλέον βελτίωση του σωματιδίου μετά από την βελτίωση από κάποιον άλλο πλήρη τελεστή.

6.2.3 Ολοκλήρωση φάσης βελτιστοποίησης

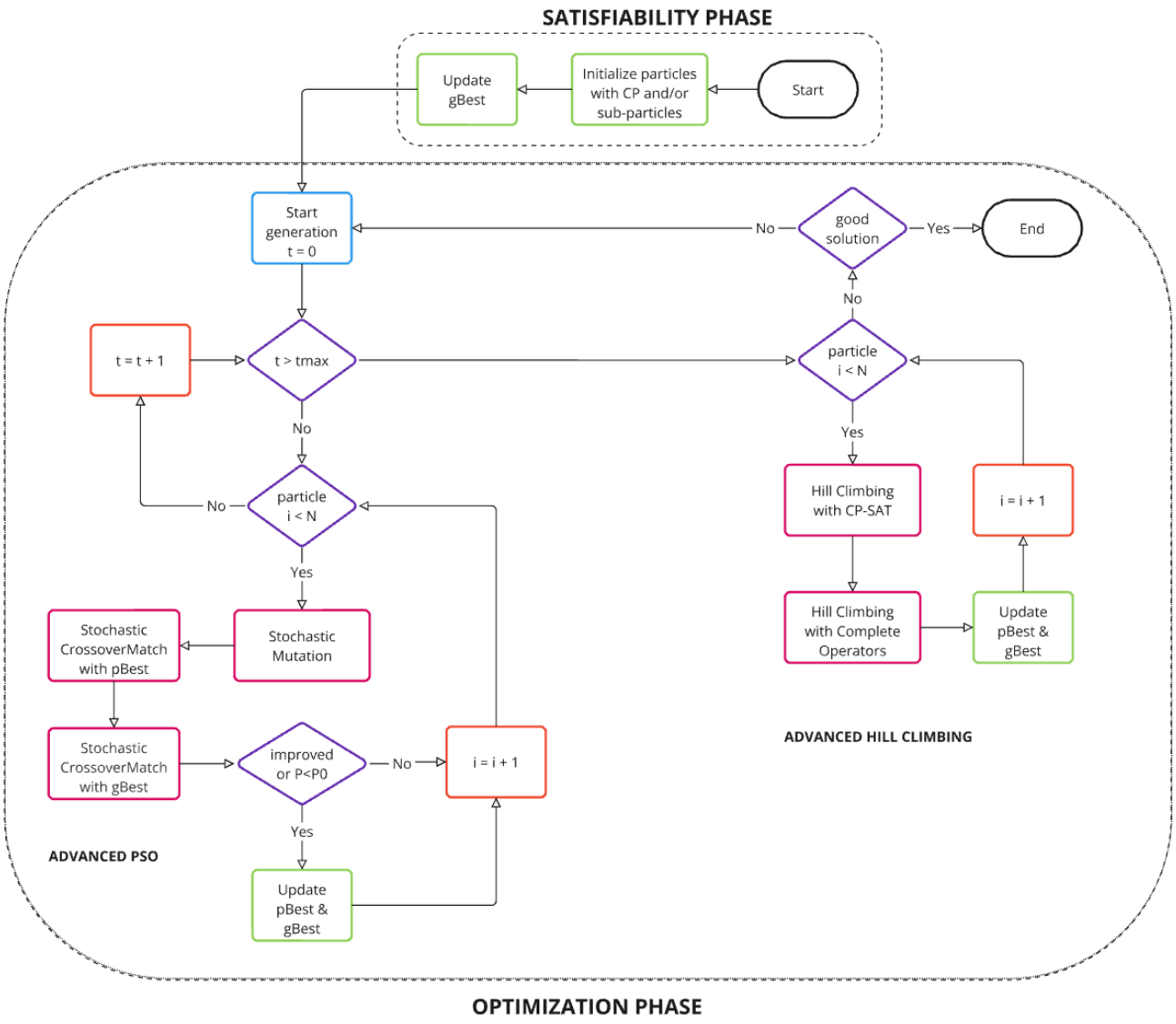
Η φάση βελτιστοποίησης ολοκληρώνεται με την ενημέρωση της μνήμης του σμήνους. Έπειτα ελέγχουμε αν έχει επιτευχθεί μία ικανοποιητική λύση. Αν αυτό συμβαίνει, τότε ο αλγόριθμος CP-PSO ολοκληρώνεται επιστρέφοντας στην έξοδό του το σωματίδιο *gBest* του

σμήνους. Διαφορετικά, εισάγει τα βελτιωμένα σωματίδια ξανά στον μηχανισμό Advanced PSO και επαναλαμβάνει την διαδικασία.

Στο Σχήμα 21 φαίνεται το διάγραμμα ροής του υβριδικού αλγορίθμου CP-PSO.



Σχήμα 20: Hill Climbing με πλήρεις τελεστές



Σχήμα 21: Ο υβριδικός αλγόριθμος CP-PSO

□

7 Υλοποίηση αλγορίθμου CP-PSO

Ο αλγόριθμος CP-PSO υλοποιήθηκε σε γλώσσα προγραμματισμού Python. Προκειμένου να επιτευχθεί η μέγιστη απόδοση του αλγορίθμου με πλήρη χρήση των υπολογιστικών πόρων του συστήματος, ο αλγόριθμος χρησιμοποιεί τεχνικές παράλληλου προγραμματισμού.

Στην υλοποίηση του CP-PSO, κάθε σωματίδιο (εφόσον το υπολογιστικό σύστημα το επιτρέπει) κάνει αποκλειστική χρήση ενός λογικού πυρήνα (vCPU) του υπολογιστικού συστήματος και ξεκινά να μεταλλάσσεται επαναληπτικά μετρώντας την δική του γενιά (η οποία στην κλάση του σωματιδίου αναφέρεται ως *ζωή* του σωματιδίου). Κάθε σωματίδιο μεταλλάσσεται αντλώντας πληροφορία όχι μόνον από την δική του μνήμη, αλλά και από την μνήμη του σμήνους (Global PSO).

Μετά από κάθε μετάλλαξή του, το σωματίδιο ενημερώνει την μνήμη του και με την βοήθεια σημαφόρων «κλειδώνει» την πρόσβαση στην μνήμη του σμήνους (*gBest*) και την ενημερώνει, εφόσον απαιτείται. Με αυτό τον τρόπο, στην επόμενη γενιά τους, όλα τα σωματίδια έχουν πρόσβαση στην νέα, ενημερωμένη μνήμη του σμήνους.

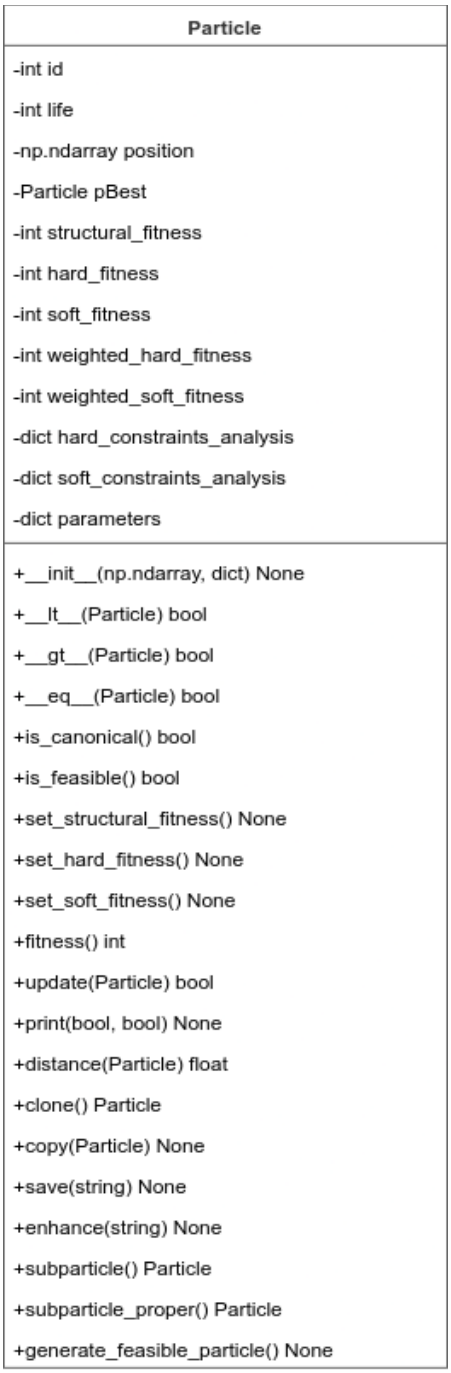
7.1 Κλάσεις

7.1.1 Κλάση Particle

Για τις ανάγκες της υλοποίησης δημιουργήθηκαν δύο βασικές κλάσεις: η κλάση Particle και η κλάση Swarm. Η κλάση Particle (Σχήμα 22) χρησιμοποιείται για να αντιπροσωπεύσει κάθε σωματίδιο του σμήνους. Η υλοποίηση της κλάσης Particle σε γλώσσα Python δίνεται στον Πίνακα 5. Οι μεταβλητές στιγμιοτύπου της κλάσης αυτής είναι οι εξής:

- *id*: η ταυτότητα του σωματιδίου.
- *life*: η ζωή (ή γενιά) του σωματιδίου που αυξάνεται κατά 1 σε κάθε επανάληψη.
- *position*: η θέση του σωματιδίου στον χώρο αναζήτησης. Η θέση δηλώνεται με έναν πίνακα 2 διαστάσεων που χρησιμοποιεί την κωδικοποίηση $x_{i,s}$ και περιγράφει ένα πλήρες ωρολόγιο πρόγραμμα κατά την χρονική στιγμή της ζωής του σωματιδίου.
- *pbest:* η μνήμη του σωματιδίου στην οποία αποθηκεύεται η καλύτερη εκδοχή του σωματιδίου που έχει επιτευχθεί μέχρι στιγμής.

- *structural_fitness*: το συνολικό κόστος παραβίασης των δομικών περιορισμών του σωματιδίου.
- *hard_fitness*: το συνολικό κόστος παραβίασης των ισχυρών περιορισμών του σωματιδίου.
- *soft_fitness*: το συνολικό κόστος παραβίασης των ασθενών περιορισμών του σωματιδίου.
- *weighted_hard_constaint*: το συνολικό κόστος παραβίασης των ισχυρών περιορισμών του σωματιδίου μετά την πολλαπλασιαστική προσαύξηση του πραγματικού κόστους με έναν συντελεστή ειδικού βάρους.
- *weighted_soft_constraint*: το συνολικό κόστος παραβίασης των ασθενών περιορισμών του σωματιδίου μετά την πολλαπλασιαστική προσαύξηση του πραγματικού κόστους με έναν συντελεστή ειδικού βάρους.
- *hard_constraints_analysis*: μία μεταβλητή τύπου dictionary που αποθηκεύει το κόστος ανά ισχυρό περιορισμό.
- *soft_constraints_analysis*: μία μεταβλητή τύπου dictionary που αποθηκεύει το κόστος ανά ασθενές περιορισμό.
- *parameters*: οι παράμετροι του προβλήματος



Σχήμα 22: Διάγραμμα κλάσης Particle

Οι μέθοδοι στιγμιοτύπου της κλάσης Particle είναι οι εξής:

- *__init__*: ο constructor της κλάσης που δέχεται 2 ορίσματα: έναν πίνακα τύπου $x_{i,s}$ και τις παραμέτρους του προβλήματος και δημιουργεί ένα αντικείμενο τύπου *Particle* αρχικοποιώντας τις μεταβλητές στιγμιοτύπου του.
- *is_canonical*: ελέγχει αν το σωματίδιο είναι κανονικό.
- *is_feasible*: ελέγχει αν το σωματίδιο είναι εφικτό.
- *__lt__*: ορίζει την σχέση $\prec$ μεταξύ δύο σωματιδίων σύμφωνα με το μέτρο σύγκρισης που έχει οριστεί στο κεφάλαιο 5.4.
- *__gt__*: ορίζει την σχέση $\succ$ μεταξύ δύο σωματιδίων σύμφωνα με το μέτρο σύγκρισης που έχει οριστεί στο κεφάλαιο 5.4.
- *__eq__*: ορίζει την ισότητα $\equiv$ μεταξύ δύο σωματιδίων σύμφωνα με το μέτρο σύγκρισης που έχει οριστεί στο κεφάλαιο 5.4.
- *set_structural_fitness*: υπολογίζει το συνολικό κόστος της παραβίασης των δομικών περιορισμών του σωματιδίου.
- *set_hard_fitness*: υπολογίζει το συνολικό κόστος της παραβίασης των ισχυρών περιορισμών του σωματιδίου.
- *set_soft_fitness*: υπολογίζει το συνολικό κόστος της παραβίασης των ασθενών περιορισμών του σωματιδίου.
- *fitness*: υπολογίζει το συνολικό κόστος παραβίασης όλων των περιορισμών του σωματιδίου.
- *update*: ενημερώνει την θέση του σωματιδίου κάθε χρονική στιγμή. Δέχεται ως όρισμα το σωματίδιο *gBest*.
- *print*: εκτυπώνει το συνολικό κόστος, την ανάλυση κόστους ανά τύπο και κατηγορία περιορισμού, και το ωρολόγιο πρόγραμμα που αντιστοιχεί στην θέση του σωματιδίου.
- *distance*: υπολογίζει την απόσταση ανάμεσα στο τρέχον σωματίδιο και σε ένα άλλο σωματίδιο, το οποίο δέχεται ως όρισμα.
- *clone*: δημιουργεί έναν κλώνο του τρέχοντος σωματιδίου.
- *copy*: αντιγράφει από ένα σωματίδιο την θέση και το κόστος στο τρέχον σωματίδιο.

- *save*: αποθηκεύει την θέση του σωματιδίο σε ένα τοπικό αρχείο δεδομένων.
- *enhance*: βελτιώνει το σωματίδιο. Δέχεται ως όρισμα μία αλφαριθμητική παράμετρο που καθορίζει την μέθοδο βελτίωσης που θα χρησιμοποιηθεί (CP-SAT, πλήρεις τελεστές).
- *subparticle*: παράγει ένα υποσωματίδιο από το τρέχον σωματίδιο.
- *subparticle_proper*: παράγει ένα γνήσιο υποσωματίδιο από το τρέχον σωματίδιο.
- *generate_feasible_particle*: παράγει ένα εφικτό σωματίδιο από ένα κανονικό σωματίδιο χρησιμοποιώντας υποσωματίδια.

```
class Particle:

    def __init__(self, x_is: np.ndarray, instance_and_parameters: dict) -> None:
        """ Constructor of the particle. """
        self.id: int = 0
        self.life: int = 0
        self.parameters: dict = instance_and_parameters
        self.position: np.ndarray = copy.deepcopy(x_is)
        self.structural_fitness: int = 0
        self.hard_fitness: int = 0
        self.soft_fitness: int = 0
        self.weighted_hard_fitness: int = 0
        self.weighted_soft_fitness: int = 0
        self.hard_constraints_analysis: dict = {}
        self.soft_constraints_analysis: dict = {}
        self.set_structural_fitness()
        self.set_hard_fitness()
        self.set_soft_fitness()
        self.pBest: "Particle" = copy.deepcopy(self)


    def is_canonical(self) -> bool:
        """ A particle is canonical if its structural fitness is zero. """
        return self.structural_fitness == 0


    def is_feasible(self) -> bool:
        """ A particle is feasible if it is canonical and its hard fitness is zero. """
        return self.is_canonical() and self.hard_fitness == 0


    def __lt__(self, other: "Particle") -> bool:
        """
        A < B means A is better than B.

        A feasible particle is always better than a canonical which is always
        better than a simple particle.
        """
        if self.is_feasible():
            if other.is_feasible():
                return self.soft_fitness < other.soft_fitness
            else:
                return True
```

```
44            else:
45                if other.is_feasible():
46                    return False
47                else:
48                    if self.is_canonical():
49                        if other.is_canonical():
50                            return self.hard_fitness < other.hard_fitness
51                        else:
52                            return True
53                    else:
54                        if other.is_canonical():
55                            return False
56                        else:
57                            return self.structural_fitness < other.structural_fitness
58
59
60        def __gt__(self, other: "Particle") -> bool:
61            """
62            A > B means A is worse than B.
63
64            A feasible particle is always better than a canonical which is always
65            better than a simple particle.
66            """
67            return other < self
68
69
70        def __eq__(self, other: "Particle") -> bool:
71            """ A particle A is equal to B, if their timetables are identical. """
72            return np.array_equal(self.position, other.position)
73
74
75        def set_structural_fitness(self) -> None:
76            """
77            Sets the structural fitness of the particle.
78            Structural constraints are:
79            1. Rule 0
80            2. RR Rule
81            3. Phase Rule
82            4. Compactness (checked and applied when reading parameters)
83            5. Rule 1 determined by particle generator
84            """
85            penalty: int = 0
86            g_ijs = convert_xis_to_g(self.position)
87            # Check that each team plays at most once per time slot. If not, apply penalty
88            if not is_rule0_satisfied(g_ijs):
89                penalty += self.parameters["weights"]["structural"]
90            # Check the kRR restriction
91            if not is_RR_satisfied(g_ijs, self.parameters["RR"], 0, self.parameters['slots_num']):
92                penalty += self.parameters["weights"]["structural"]
93            # Check Phase Rule
94            if self.parameters["gamemode"] == "P":
95                if not is_phased(g_ijs, self.parameters):
96                    penalty += self.parameters["weights"]["structural"]
97            self.structural_fitness = penalty
98
99
100       def set_hard_fitness(self) -> None:
101           """ Sets the hard fitness and the hard constraints analysis of the particle. """
102           self.hard_constraints_analysis = {"CA1": 0,
103                                             "CA2": 0,
104                                             "CA3": 0,
105                                             "CA4": 0,
106                                             "CA5": 0,
107                                             "GA1": 0,
108                                             "GA2": 0,
```

```
                                        "BR1": 0,
                                        "BR2": 0,
                                        "BR3": 0,
                                        "BR4": 0,
                                        "FA1": 0,
                                        "FA2": 0,
                                        "FA3": 0,
                                        "FA4": 0,
                                        "FA5": 0,
                                        "FA6": 0,
                                        "SE1": 0,
                                        "SE2": 0,
                                        }
            g_ijs = convert_xis_to_g(self.position)
            real_hard_fitness: int = 0
            for key in self.parameters["constraints"]:
                if self.parameters["constraints"][key]["ctype"] == "HARD":
                    real_cost, _ = get_constraint_cost(key, self.parameters, g_ijs)
                    self.hard_constraints_analysis[self.parameters["constraints"][key]["category"]] += real_cost
                    real_hard_fitness += real_cost
            self.hard_fitness = real_hard_fitness
            self.weighted_hard_fitness = self.parameters["weights"]["hard"] * real_hard_fitness


        def set_soft_fitness(self) -> None:
            """ Sets the soft fitness and the soft constraints analysis of the particle. """
            self.soft_constraints_analysis = {"CA1": 0,
                                        "CA2": 0,
                                        "CA3": 0,
                                        "CA4": 0,
                                        "CA5": 0,
                                        "GA1": 0,
                                        "GA2": 0,
                                        "BR1": 0,
                                        "BR2": 0,
                                        "BR3": 0,
                                        "BR4": 0,
                                        "FA1": 0,
                                        "FA2": 0,
                                        "FA3": 0,
                                        "FA4": 0,
                                        "FA5": 0,
                                        "FA6": 0,
                                        "SE1": 0,
                                        "SE2": 0,
                                        }
            g_ijs = convert_xis_to_g(self.position)
            real_soft_fitness: int = 0
            weighted_soft_fitness: int = 0
            for key in self.parameters["constraints"]:
                if self.parameters["constraints"][key]["ctype"] == "SOFT":
                    real_cost, weighted_cost = get_constraint_cost(key, self.parameters, g_ijs)
                    self.soft_constraints_analysis[self.parameters["constraints"][key]["category"]] += real_cost
                    real_soft_fitness += real_cost
                    weighted_soft_fitness += weighted_cost
            self.soft_fitness = real_soft_fitness
            self.weighted_soft_fitness = weighted_soft_fitness


        def fitness(self) -> int:
            """ Gets the fitness of the particle. """
            return self.structural_fitness + self.hard_fitness + self.soft_fitness
```

```
173         def update(self, gbest: "Particle") -> bool:
174             """ Updates the particle. """
175             isPhased: bool = True if self.parameters["gamemode"] == "P" else False
176             k: float = self.parameters["k"]
177             # Termination condition
178             if self.life >= self.parameters["max_age"]:
179                 return False
180             c_particle: Particle = self.clone()
181             # Stochastic Mutation
182             if random.uniform(0,1) <= self.parameters["inertia_probability"]:
183                 c_particle = mutate_random(c_particle)
184             # Stochastic Crossover with pBest
185             if random.uniform(0,1) <= self.parameters["cognitive_probability"]:
186                 c_particle = crossover_match_random(c_particle, self.pBest, isPhased)
187             # Stochastic Crossover with gBest
188             if random.uniform(0,1) <= self.parameters["social_probability"]:
189                 c_particle = crossover_match_random(c_particle, gbest, isPhased)
190             # Metropolis Criterion: Accept transformation if it's better or with probability
191             weighted_fitness: int = c_particle.weighted_hard_fitness + c_particle.weighted_soft_fitness
192             probability: float = math.exp(-k * weighted_fitness/(self.parameters["max_age"] - self.life))
193             if c_particle < self or random.uniform(0,1) < probability:
194                 self.copy(c_particle)
195                 fitness_str: str = f"{self.structural_fitness} + {self.hard_fitness} + {self.soft_fitness}"
196                 print(f"{self.id:<10}{self.life:<10}{fitness_str:<20}")
197             return True
198 
199 
200         def print(self, with_timetable: bool = False, with_constraint_analysis: bool = False) -> None:
201             """
202             Prints
203             - the fitness of the particle,
204             - the position (timetable) of the particle,
205             - the analysis of its constraints.
206             """
207             fitness_str: str = f"{self.structural_fitness} + {self.hard_fitness} + {self.soft_fitness}"
208             print(f"{self.id:<10}{self.life:<10}{fitness_str:<20}")
209             # Separate satisfied from violated hard constraints
210             satisfied: list = []
211             violated: list = []
212             g_ijs = convert_xis_to_g(self.position)
213             for key in self.parameters["constraints"]:
214                 if self.parameters["constraints"][key]["ctype"] == "HARD":
215                     real_cost, _ = get_constraint_cost(key, self.parameters, g_ijs)
216                     if real_cost == 0:
217                         satisfied.append(key)
218                     else:
219                         violated.append(key)
220             # Print hard and soft contraints analysis
221             if with_constraint_analysis:
222                 print("Hard constraints analysis")
223                 print(f"{len(satisfied) + len(violated)} = {len(satisfied)} satisfied + {len(violated)} violated")
224                 for key in self.hard_constraints_analysis:
225                     value = self.hard_constraints_analysis.get(key)
226                     if value != 0:
227                         print(f"{key}: {value}")
228                 print("Soft constraints analysis")
229                 for key in self.soft_constraints_analysis:
230                     value = self.soft_constraints_analysis.get(key)
231                     if value != 0:
232                         print(f"{key}: {value}")
233             # Print timetable
234             if with_timetable:
235                 teams_num: int = len(self.position)
236                 slots_num: int = len(self.position[0])
237                 g: np.ndarray = convert_xis_to_g(self.position)
```

```
            for s in range(slots_num):
                print(f"\nslot {s}\t", end="")
                for i in range(teams_num):
                    for j in range(teams_num):
                        if g[i][j][s] != 0:
                            print(f"({i},{j})\t", end="")
                print("")


    def distance(self, p: "Particle") -> float:
        """
        Measure the distance of two particles of the same problem.

        If two particles are equal, then d = 0.
        If they differ in n games, then d = n / gamesNum.
        If they are totally different, then d = gamesNum / gamesNum.
        The metric returns d normalized in the interval [0, 1].

        Arguments:
        - p: Particle to be compare with the current particle

        Returns:
        - distance: float
        """
        # If both particles have the same dimensions,
        if self.parameters["teams_num"] == p.parameters["teams_num"] and
                                                self.parameters["slots_num"] == p.parameters["slots_num"]:
            # convert each position to x_s format
            x1_s = convert_xis_to_xs(self.position)
            x2_s = convert_xis_to_xs(p.position)
            teams_num: int = self.parameters["teams_num"]
            slots_num: int = self.parameters["slots_num"]
            games_num: int = teams_num * (teams_num - 1)
            differences: int = 0
            # and compare each game at each slot of the first particle with all games
            # at the same slot of the other particle
            for s in range(slots_num):
                for m in range(teams_num // 2):
                    isDifferent = True
                    for k in range(teams_num // 2):
                        if x1_s[s][m] == x2_s[s][k]:
                            isDifferent = False
                    # Collect the differences
                    if isDifferent:
                        differences += 1
            d: float = differences / games_num
            return d
        else:
            return -1


    def clone(self) -> "Particle":
        """ Generate a clone of the particle. """
        c_particle: "Particle" = copy.deepcopy(self)
        return c_particle


    def copy(self, particle: "Particle") -> None:
        """
        Transfer position and fitness information to the particle
        and update its pBest

        Arguments
        - particle: the source of information to copy from.
        """
```

```
303             self.position = copy.deepcopy(particle.position)
304             self.structural_fitness = particle.structural_fitness
305             self.hard_fitness = particle.hard_fitness
306             self.weighted_hard_fitness = particle.weighted_hard_fitness
307             self.soft_fitness = particle.soft_fitness
308             self.weighted_soft_fitness = particle.weighted_soft_fitness
309             self.soft_constraints_analysis = particle.soft_constraints_analysis
310             # Update pbest
311             if self < self.pBest:
312                 self.pBest = self.clone()
313
314
315         def save(self) -> None:
316             """ Save the particle in a local file. """
317             np.save(f"{self.structural_fitness}+{self.hard_fitness}+{self.soft_fitness}", self.position)
318
319
320         def enhance(self, method: str) -> None:
321             """
322             Enhance the particle.
323
324             Arguments:
325             - Method: the method of enhancement ("CP-SAT", "COMPLETE-OPERATORS")
326             """
327             match method:
328                 case "CP-SAT":
329                     # Hill Climbing with CP-SAT.
330                     q: Particle = cpsat_enhancer(self)
331                     while q < self:
332                         q.save()
333                         self.copy(q)
334                         q = cpsat_enhancer(self)
335                 case "COMPLETE-OPERATORS":
336                     # Hill Climbing with complete operators.
337                     isPhased: bool = True if self.parameters["gamemode"] == "P" else False
338                     p: Particle = self.clone()
339                     q: Particle
340                     restarting_flag: bool = False
341                     while True:
342                         # Swap homes
343                         print(f"ID:{p.id} Enhancing with swap homes")
344                         q = swap_homes_complete_async(p)
345                         q.print()
346                         while q < p:
347                             q.save()
348                             p.copy(q)
349                             q = swap_homes_complete_async(p)
350                             q.print()
351                         # Swap rounds
352                         print(f"ID:{p.id} Enhancing with swap rounds")
353                         q = swap_rounds_complete_async(p, isPhased)
354                         q.print()
355                         while q < p:
356                             q.save()
357                             p.copy(q)
358                             q = swap_rounds_complete_async(p, isPhased)
359                             q.print()
360                             restarting_flag = True
361                         if restarting_flag:
362                             restarting_flag = False
363                             continue
364                         # Swap teams
365                         print(f"ID:{p.id} Enhancing with swap teams")
366                         q = swap_teams_complete_async(p)
367                         q.print()
```

```
368                 while q < p:
369                     q.save()
370                     p.copy(q)
371                     q = swap_teams_complete_async(p)
372                     q.print()
373                     restarting_flag = True
374                 if restarting_flag:
375                     restarting_flag = False
376                     continue
377                 # Swap opponents
378                 print(f"ID:{p.id} Enhancing with swap opponents")
379                 q = swap_opponents_complete_async(p)
380                 q.print()
381                 while q < p:
382                     q.save()
383                     p.copy(q)
384                     q = swap_opponents_complete_async(p)
385                     q.print()
386                     restarting_flag = True
387                 if restarting_flag:
388                     restarting_flag = False
389                     continue
390                 # Rotate teams
391                 print(f"ID:{p.id} Enhancing with rotate teams")
392                 q = rotate_teams_complete_async(p)
393                 q.print()
394                 while q < p:
395                     q.save()
396                     p.copy(q)
397                     q = rotate_teams_complete_async(p)
398                     q.print()
399                     restarting_flag = True
400                 if restarting_flag:
401                     restarting_flag = False
402                     continue
403                 # Swap matches
404                 print(f"ID:{p.id} Enhancing with swap matches")
405                 q = swap_matches_complete_async(p, isPhased)
406                 q.print()
407                 while q < p:
408                     q.save()
409                     p.copy(q)
410                     q = swap_matches_complete_async(p, isPhased)
411                     q.print()
412                     restarting_flag = True
413                 if restarting_flag:
414                     restarting_flag = False
415                     continue
416                 # Crossover match
417                 print(f"ID:{p.id} Enhancing with crossover match")
418                 q = crossover_match_complete_async(p, p, isPhased)
419                 q.print()
420                 while q < p:
421                     q.save()
422                     p.copy(q)
423                     q = crossover_match_complete_async(p, p, isPhased)
424                     q.print()
425                     restarting_flag = True
426                 if restarting_flag:
427                     restarting_flag = False
428                     continue
429                 # Swap matches & crossover
430                 print(f"ID:{p.id} Enhancing with swap matches crossover")
431                 q = swap_matches_crossover_complete_async(p, isPhased)
432                 q.print()
```

```
                    while q < p:
                        q.save()
                        p.copy(q)
                        q = swap_matches_crossover_complete_async(p, isPhased)
                        q.print()
                        restarting_flag = True
                    if restarting_flag:
                        restarting_flag = False
                        continue
                    else:
                        break

                # Update self particle
                self.copy(p)
            case _:
                raise ValueError("Not a valid argument.")


    def subparticle(self) -> "Particle":
        """
        Produce a sub-particle.

        Arguments:
        - Particle

        Returns:
        - Particle
        """
        # Copy initial particle
        parameters: dict = copy.deepcopy(self.parameters)
        timetable: np.ndarray = self.position
        satisfied: list = []
        violated: list = []
        violated_cost: list = []
        g_ijs = convert_xis_to_g(timetable)
        # Separate satisfied from violated hard constraints
        for key in parameters["constraints"]:
            if parameters["constraints"][key]["ctype"] == "HARD":
                real_cost, _ = get_constraint_cost(key, parameters, g_ijs)
                if real_cost == 0:
                    satisfied.append(key)
                else:
                    violated.append(key)
                    violated_cost.append(real_cost)
        # Remove all soft constraints
        soft_constraints: list = []
        for key in parameters["constraints"]:
            if parameters["constraints"][key]["ctype"] == "SOFT":
                soft_constraints.append(key)
        for key in soft_constraints:
            del parameters["constraints"][key]
        # Keep all satisfied hard constraints.
        # Convert all violated hard constraints into SOFT
        for key in violated:
            parameters["constraints"][key]["ctype"] = "SOFT"
        return Particle(timetable, parameters)


    def subparticle_proper(self) -> "Particle":
        """
        Produces a proper sub-particle.

        Arguments:
        - Particle
```

```
497             Returns:
498             - Particle
499             """
500             parameters: dict = copy.deepcopy(self.parameters)
501             satisfied: list = []
502             violated: list = []
503             violated_cost: list = []
504             g_ijs = convert_xis_to_g(self.position)
505             # Separate satisfied from violated hard constraints
506             for key in parameters["constraints"]:
507                 if parameters["constraints"][key]["ctype"] == "HARD":
508                     real_cost, _ = get_constraint_cost(key, parameters, g_ijs)
509                     if real_cost == 0:
510                         satisfied.append(key)
511                     else:
512                         violated.append(key)
513                         violated_cost.append(real_cost)
514             # Remove all soft constraints
515             soft_constraints: list = []
516             for key in parameters["constraints"]:
517                 if parameters["constraints"][key]["ctype"] == "SOFT":
518                     soft_constraints.append(key)
519             for key in soft_constraints:
520                 del parameters["constraints"][key]
521             # Keep all satisfied hard constraints.
522             # Remove all but "fraction" violated hard constraints and convert them to SOFT
523             fraction: int = parameters["subparticle_fraction"]
524             for key in violated[fraction:]:
525                 del parameters["constraints"][key]
526             for key in violated[:fraction]:
527                 parameters["constraints"][key]["ctype"] = "SOFT"
528             # for key in violated[:-fraction]:
529             #     del parameters["constraints"][key]
530             # for key in violated[-fraction:]:
531             #     parameters["constraints"][key]["ctype"] = "SOFT"
532             return Particle(self.position, parameters)
533
534
535         def generate_feasible_particle(self, proper_subparticles: bool) -> None:
536             """
537             Generates a feasible particle from a canonical particle with sub-particles.
538
539             Arguments:
540             - p: Particle
541             - proper_subparticles: bool (True for using proper sub-particles)
542
543             """
544             if self.is_feasible():
545                 return
546             # Derive sub-particle
547             if proper_subparticles:
548                 q: Particle = self.subparticle_proper()
549             else:
550                 q: Particle = self.subparticle()
551             print("Sub-particle")
552             # Reversed particle from sub-particle
553             q_c: Particle = Particle(q.position, self.parameters)
554             q.print()
555             q_c.print(False, True)
556             # Enhance sub-particle leading to a new sub-particle
557             r: Particle = cpsat_enhancer(q)
558             # Reversed particle from enhanced sub-particle
559             r_c: Particle = Particle(r.position, self.parameters)
560             print("Enhanced sub-particle")
561             r.print()
```

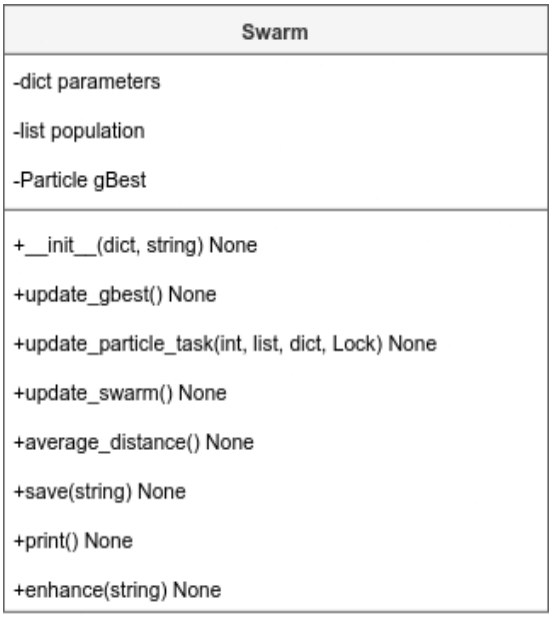
```
562             r_c.print(False, True)
563             r_c.save()
564             # Continue as long as r_c becomes feasible
565             while not r_c.is_feasible():
566                 # if r is not better than q, then double runtime
567                 if not r < q:
568                     r_c.parameters["cpsat"]["max_runtime"] *= 2
569                     r.parameters["cpsat"]["max_runtime"] *= 2
570                 # if r becomes totally zero, then derive new sub-particle from r_c.
571                 if r.soft_fitness == 0:
572                     # Derive new sub-particle
573                     if proper_subparticles:
574                         q = r_c.subparticle_proper()
575                     else:
576                         q = r_c.subparticle()
577                     print("Sub-particle")
578                     q_c: Particle = Particle(q.position, self.parameters)
579                     q.print()
580                     q_c.print(False, True)
581                 else:
582                     # Otherwise, keep the same sub-particle and improve it
583                     q = r.clone()
584                 # Enhance sub-particle
585                 r = cpsat_enhancer(q)
586                 r_c = Particle(r.position, self.parameters)
587                 print("Enhanced sub-particle")
588                 r.print()
589                 r_c.print(False, True)
590                 r_c.save()
591             self.copy(r_c)
```

Πίνακας 5: Η κλάση Particle

7.1.2 Κλάση Swarm

Η κλάση Swarm (Σχήμα 23) αντιπροσωπεύει το σμήνος στο σύνολό του. Οι μεταβλητές στιγμιοτύπου της κλάσης είναι οι εξής:

- *parameters*: οι παράμετροι του προβλήματος
- *population*: μία λίστα που αποτελείται από τα σωματίδια του σμήνους.
- *gbest*: ένα αντικείμενο τύπου Particle που αποτελεί την μνήμη του σμήνους και αποθηκεύει το καλύτερο σωματίδιο που έχει βρεθεί μέχρι στιγμής.



Σχήμα 23: Διάγραμμα κλάσης Swarm

Οι μέθοδοι στιγμιοτύπου της κλάσης Swarm είναι οι εξής:

- *__init__*: ο constructor της κλάσης που δημιουργεί το αντικείμενο και δέχεται ως όρισμα δύο παραμέτρους: τις παραμέτρους του προβλήματος και ένα αρχείο δεδομένων από το οποίο διαβάζει τα σωματίδια του σμήνους.
- *update_gbest*: ενημερώνει το σωματίδιο *gBest* του σμήνους.
- *update_swarm*: ενημερώνει τα σωματίδια του σμήνους σε κάθε γενιά. Η μέθοδος αυτή χρησιμοποιεί τεχνικές παράλληλου προγραμματισμού και σε συνεργασία με την βοηθητική μέθοδο *update_particle_task* ξεκινάει μία διεργασία (*task*) για κάθε σωματίδιο του σμήνους, η οποία δεσμεύει έναν υπολογιστικό πόρο (*vCPU*) του συστήματος και εκτελείται έως ότου συμπληρωθεί η μέγιστη ηλικία του σωματιδίου.
- *average_distance*: υπολογίζει την μέση απόσταση των σωματιδίων του σμήνους. Για τον υπολογισμό χρησιμοποιεί την μετρική της απόστασης δύο σωματιδίων (σχέση 59) υπολογίζοντας την απόσταση κάθε σωματιδίου με κάθε άλλο και επιστρέφει την μέση απόσταση και την τυπική της απόκλιση.
- *save*: αποθηκεύει τα ωρολόγια προγράμματα του σμήνους σε ένα τοπικό αρχείο.
- *print*: εκτυπώνει την σύνθεση του σμήνους (ID σωματιδίου, ζωή σωματιδίου και αναλυτικό κόστος σωματιδίου).
- *enhance*: βελτιώνει κάθε σωματίδιο του σμήνους καλώντας την αντίστοιχη μέθοδο βελτίωσης της κλάσης Particle.

Η υλοποίηση της κλάσης Swarm σε γλώσσα Python δίνεται στον Πίνακα 6.

```
1  class Swarm:
2
3      def __init__(self, parameters: dict, fromFile: str) -> None:
4          """ Constructor of the swarm from a datafile. """
5          self.parameters: dict = parameters
6          # Read timetables from the datafile, sort them and keep
7          # the first particle_num of them.
8          with open(fromFile, "rb") as f:
9              timetables: np.ndarray = np.load(f, allow_pickle=True)
10         particles: list[Particle] = []
11         for id, tb in enumerate(timetables):
12             p = Particle(tb, parameters)
13             p.id = id
14             particles.append(p)
15         sorted_particles: list = sorted(particles, key=lambda p: p.fitness())
16         self.population: list = sorted_particles[:parameters["particles_num"]]
```

```
17          # Initialize gBest particle
18          self.gBest: Particle = self.population[0].clone()
19
20
21      def update_gbest(self) -> None:
22          """ Update the gBest particle of the swarm. """
23          particle: Particle
24          for particle in self.population:
25              if particle.pBest < self.gBest:
26                  self.gBest = particle.pBest.clone()
27          print(f"gBest Particle:", end="  ")
28          self.gBest.print()
29
30
31      def update_particle_task(self, p_index, share_population, shared_dict, lock) -> None:
32          """ Function for parallel updating task. """
33          p: Particle = share_population[p_index]
34          # While lowest temperature is not reached, update the particle and gBest
35          while p.update(shared_dict['gBest']):
36              # Use lock semaphore to prevent any race conditions and conflicts with other particles
37              with lock:
38                  # Read current gBest and update it if p is better
39                  if p < shared_dict['gBest']:
40                      shared_dict['gBest'] = p.clone()
41                      print("current gbest»»»»»»»»»»»»»»»»»»»»»»»»»»»»", end=" ")
42                      shared_dict['gBest'].print()
43                      p.save()
44              # Increase particle's life
45              p.life += 1
46          # If lowest temperature is reached, then restart particle's life
47          p.life = 0
48          with lock:
49              share_population[p_index] = p
50
51
52      def update_swarm(self) -> None:
53          """ Update the particles of the swarm. Parallel programming is applied. """
54          print(f"{'ID':<10}{'Life':<10}{'Fitness':<20}")
55          with Manager() as manager:
56              shared_population = manager.list(self.population)
57              shared_dict = manager.dict({'gBest': self.gBest})
58              lock = manager.Lock()
59              processes = [Process(target=self.update_particle_task,
60                      args=(i, shared_population, shared_dict, lock)) for i in range(len(self.population))]
61              # Submit processes into the pool
62              for process in processes:
63                  process.start()
64              # Join processes
65              for process in processes:
66                  process.join()
67              # Update the main population and gBest with the final values
68              self.gBest = shared_dict['gBest']
69              self.population = list(shared_population)
70
71
72      def average_distance(self) -> None:
73          """ Measure the average distance of the particles of the swarm. """
74          # Compare each particle with all other
75          distances: list[float] = []
76          for i, p in enumerate(self.population):
77              for j, q in enumerate(self.population):
78                  if i < j:
79                      distances.append(p.distance(q))
80          # Estimate average distance and standard deviation
81          average: float = float(np.average(distances)) * 100
```

```
82          standard_deviation: float = float(np.std(distances)) * 100
83          print(f"\nParticles Average Distance = ({average:.2f} ± {standard_deviation:.2f})%")
84
85
86      def save(self, file_name = "swarm_timetables") -> None:
87          """ Save the swarm timetables in a datafile. """
88          timetables_list: list = []
89          for particle in self.population:
90              timetables_list.append(particle.position)
91          np.save(file_name, timetables_list)
92
93
94      def print(self) -> None:
95          """ Print a list of the particles of the swarm. """
96          print(f"Swarm Particles")
97          print(f"{'ID':<10}{'Life':<10}{'Fitness':<20}{'pbest':<20}{'gbest':<20}")
98          for p in self.population:
99              fitness: str = f"{p.structural_fitness} + {p.hard_fitness} + {p.soft_fitness}"
100             p_fitness: str = f"{p.pBest.structural_fitness} + {p.pBest.hard_fitness} + {p.pBest.soft_fitness}"
101             g_fitness: str = f"{self.gBest.structural_fitness} + {self.gBest.hard_fitness} +
102                                                                 {self.gBest.soft_fitness}"
103             print(f"{p.id:<10}{p.life:<10}{fitness:<20}{p_fitness:<20}{g_fitness:<20}")
104
105
106     def enhance(self, method: str) -> None:
107         """
108         Enhance the particles of the swarm.
109
110         Arguments:
111         - Method: the method of enhancement ("CP-SAT", "COMPLETE-OPERATORS")
112
113         Returns:
114         - None
115         """
116         p: Particle
116         for p in self.population:
117             print(f"Enhancing {p.id}|{p.structural_fitness} + {p.hard_fitness} + {p.soft_fitness} with {method}")
118             p.enhance(method)
119             print(f"Enhanced  {p.id}|{p.structural_fitness} + {p.hard_fitness} + {p.soft_fitness} with {method}")
```

Πίνακας 6: Η κλάση Swarm

7.2 Βοηθητικές συναρτήσεις

7.2.1 Ευρετικοί τελεστές

Η υλοποίηση των 6 ευρετικών τελεστών (*swap_homes, swap_rounds, swap_teams, swap_opponents, swap_matches* και *rotate_teams*) φαίνεται στον ακόλουθο Πίνακα 7.

```
1   def swap_homes(particle: Particle, t1: int, t2: int) -> Particle:
2       """ Swap game (t1, t2) with game (t2, t1). """
3       x_ij = convert_xis_to_xij(particle.position)
4       # Swap (i,j) with (j,i)
5       s1: int = x_ij[t1][t2]
6       s2: int = x_ij[t2][t1]
7       x_ij[t1][t2] = s2
```

```
        x_ij[t2][t1] = s1
        x_is = convert_xij_to_xis(x_ij, particle.parameters['slots_num'])
        return Particle(x_is, particle.parameters)


    def swap_rounds(particle: Particle, s1: int, s2: int) -> Particle:
        """ Swap the matches of slot s1 with those of slot s2. """
        # Swap slot s1 with s2
        y_is = swap_columns(particle.position, s1, s2)
        return Particle(y_is, particle.parameters)


    def swap_teams(particle: Particle, t1: int, t2: int) -> Particle:
        """ Swap team t1 with team t2 in all matches of x_is. """
        teams_num: int = len(particle.position)
        slots_num: int = len(particle.position[0])
        # Convert x[i,s] to x[s]
        x_s: list = convert_xis_to_xs(particle.position)
        y_is: np.ndarray = np.full([teams_num, slots_num], fill_value = -1, dtype=int)
        # For each match in x[s], if i is t1, swap it with t2.
        # If i is t2, swap it with t1. Same for j.
        for s in range(slots_num):
            for match in x_s[s]:
                i: int = match[0]
                j: int = match[1]
                if i == t1:
                    i = t2
                elif i == t2:
                    i = t1
                if j == t1:
                    j = t2
                elif j == t2:
                    j = t1
                y_is[i, s] = j
        return Particle(y_is, particle.parameters)


    def swap_opponents(particle: Particle, t1: int, t2: int) -> Particle:
        """ Swap opponents of team t1 with those of team t2 in all slots of x_is. """
        teams_num: int = len(particle.position)
        slots_num: int = len(particle.position[0])
        # Convert x[i,s] to x[s]
        x_s: list = convert_xis_to_xs(particle.position)
        y_is: np.ndarray = np.full([teams_num, slots_num], fill_value = -1, dtype=int)
        opponent: list = []
        for s in range(slots_num):
            # First traverse all matches at slot s, look for t1 or t2
            # and save the opponent
            for match in x_s[s]:
                # Match (t1, t2) or (t2, t1) is omitted
                if match != (t1, t2) and match != (t2, t1):
                    i: int = match[0]
                    j: int = match[1]
                    if i == t1 or i == t2:
                        opponent.append(j)
                    elif j == t1 or j == t2:
                        opponent.append(i)
            # Then, traverse again all matches at slot s, look for t1 or t2
            # and replace the opponent with that of the other match,
            # only if len(opponent) = 2
            k: int = 0
            for match in x_s[s]:
                i: int = match[0]
                j: int = match[1]
                if len(opponent) == 2:
```

```
                if i == t1 or i == t2:
                    j = opponent[1 - k]
                    k += 1
                elif j == t1 or j == t2:
                    i = opponent[1 - k]
                    k += 1
            # Before saving the match (i, j), check if there is already in timetable.
            # If so, then schedule the rematch (j, i) instead.
            foundBefore: bool = False
            for slot in range(s):
                if y_is[i, slot] == j:
                    foundBefore = True
                    break
            if foundBefore:
                y_is[j, s] = i
            else:
                y_is[i, s] = j
        # Clear opponent array
        opponent = []
    return Particle(y_is, particle.parameters)


def swap_matches(particle: Particle, s1: int, s2: int, match_index: int) -> Particle:
    """ Swap the match with index "match_index" of slot s1 with the match of the same index at slot s2. """
    match_indexes: list = []
    # Convert x[i,s] to x[s]
    x_s: list = convert_xis_to_xs(particle.position)
    match_indexes.append(match_index)
    # Swap selected match between s1 and s2
    match1: tuple = x_s[s1][match_index]
    match2: tuple = x_s[s2][match_index]
    x_s[s1][match_index] = match2
    x_s[s2][match_index] = match1
    # Check rule 0 at s1.
    # Since all duplicates are due to the swapping between s1 and s2,
    # we check rule 0 at s1 only. If we correct s1, then s2 will be corrected as well.
    flattened: list = [team for game in x_s[s1] for team in game]
    counts: dict = {}
    for team in flattened:
        counts[team] = counts.get(team, 0) + 1
    # Find duplicates at slot s1
    duplicates: list = [team for team, count in counts.items() if count > 1]
    # While there are duplicates in slot s1,
    while len(duplicates) > 0:
        # take the first duplicate team.
        d: int = duplicates[0]
        for position, match in enumerate(x_s[s1]):
            # If duplicate team is in a match other than the match we swapped between s1 and s2, then
            if d in match and position not in match_indexes:
                # swap selected match between s1 and s2
                match1: tuple = x_s[s1][position]
                match2: tuple = x_s[s2][position]
                x_s[s1][position] = match2
                x_s[s2][position] = match1
                # add this position to memory, so you won't select it again
                match_indexes.append(position)
                # and check rule 0 at s1 again by finding duplicates in teams
                flattened: list = [team for game in x_s[s1] for team in game]
                counts: dict = {}
                for team in flattened:
                    counts[team] = counts.get(team, 0) + 1
                # Find duplicates
                duplicates = [team for team, count in counts.items() if count > 1]
                break
    y_is = convert_xs_to_xis(x_s, particle.parameters['teams_num'])
```

```
137         return Particle(y_is, particle.parameters)
138
139
140     def rotate_teams(particle: Particle, step: int) -> Particle:
141         """
142         Rotate team numbers with rotation step
143         (i.e. one position to the right if step > 0, or to the left if step < 0).
144         In that way, the operator with step < 0 is the inverse operator of that with step > 0.
145         """
146         g_ijs: np.ndarray = convert_xis_to_g(particle.position)
147         rotated_xis: np.ndarray = np.full((particle.parameters['teams_num'],
148                                            particle.parameters['slots_num']), fill_value=-1, dtype=int)
149         for i in range(particle.parameters['teams_num']):
150             for j in range(particle.parameters['teams_num']):
151                 for s in range(particle.parameters['slots_num']):
152                     if g_ijs[i, j, s] == 1:
153                         # Calculate new team numbers with rotation step
154                         new_i: int = (i + step) % particle.parameters['teams_num']
155                         new_j: int = (j + step) % particle.parameters['teams_num']
156                         # Set the game in the new timetable
157                         rotated_xis[new_i, s] = new_j
158         return Particle(rotated_xis, particle.parameters)
```

Πίνακας 7: Ευρετικοί τελεστές

Προκειμένου να υλοποιηθεί ο μηχανισμός της μετάλλαξης του σωματιδίου, είναι απαραίτητο να τυχαιοποιηθεί κάθε ευρετικός τελεστής. Αυτό γίνεται με τις ακόλουθες βοηθητικές συναρτήσεις (Πίνακας 8). Κάθε μία, όταν κληθεί, εκτελεί τον αντίστοιχο ευρετικό τελεστή με ορίσματα που επιλέγονται τυχαία (με ομοιόμορφη κατανομή) από μία δεξαμενή επιλογών.

```
1   def swap_homes_random(particle: Particle) -> Particle:
2       """ Randomly select two teams i, j where i != j and then swap game (i, j) with game (j, i). """
3       t1, t2 = get_random_sample(particle.parameters['teams_num'])
4       return swap_homes(particle, t1, t2)
5
6
7   def swap_teams_random(particle: Particle) -> Particle:
8       """ Randomly select two teams t1, t2 where t1 != t2 and swap t1 with t2 in all matches. """
9       t1, t2 = get_random_sample(particle.parameters['teams_num'])
10      return swap_teams(particle, t1, t2)
11
12
13  def swap_opponents_random(particle: Particle) -> Particle:
14      """ Randomly select teams t1, t2 where t1 != t2 and swap opponents of t1 with those of t2 in all slots. """
15      t1, t2 = get_random_sample(particle.parameters['teams_num'])
16      return swap_opponents(particle, t1, t2)
17
18
19  def swap_rounds_random(particle: Particle, isPhased: bool) -> Particle:
20      """
21      Randomly select two slots s1, s2 where s1 != s2 and swap the matches of s1 with those of s2.
22      For phased timetables, assume RR = 2.
23      """
24      # If it is phases, then swap slot s1 with slot s2
```

```
25      # only if s1, s2 belong to the same phase.
26      if isPhased:
27          # Phased case
28          # Select a phase randomly
29          phase: int = random.randint(0, 1)
30          m: int = particle.parameters['slots_num'] // 2
31          s1: int = random.randint(phase * m, (1 + phase) * m - 1)
32          s2: int = random.randint(phase * m, (1 + phase) * m - 1)
33          while s1 == s2:
34              s2 = random.randint(phase * m, (1 + phase) * m - 1)
35      else:
36          # Non-Phased case
37          # Produce random s1, s2 different from each other
38          s1, s2 = get_random_sample(particle.parameters['slots_num'])
39      # Swap slot s1 with s2
40      return swap_rounds(particle, s1, s2)
41
42
43  def swap_matches_random(particle: Particle, isPhased: bool) -> Particle:
44      """
45      Randomly select a game of slot s1 and swap it with the matching game at slot s2.
46      Then, check Rule 0 and force more swaps until Rule 0 is satisfied.
47      Assuming Phases = 2 (if it is Phased)
48      """
49      # If it is phased, select to swap from slots s1, s2 that belong to the same phase.
50      if isPhased:
51          # Phased case
52          # Select a phase randomly
53          phase: int = random.randint(0, 1)
54          m: int = particle.parameters['slots_num'] // 2
55          s1: int = random.randint(phase * m, (1 + phase) * m - 1)
56          s2: int = random.randint(phase * m, (1 + phase) * m - 1)
57          while s1 == s2:
58              s2 = random.randint(phase * m, (1 + phase) * m - 1)
59      else:
60          # Non-Phased case
61          # Produce random s1, s2 different from each other
62          s1, s2 = get_random_sample(particle.parameters['slots_num'])
63      # Convert x[i,s] to x[s]
64      x_s: list = convert_xis_to_xs(particle.position)
65      # Randomly select a match from slot s1
66      match_index: int = random.randint(0, len(x_s[s1]) - 1)
67      return swap_matches(particle, s1, s2, match_index)
68
69
70  def rotate_teams_random(particle: Particle) -> Particle:
71      """ Rotate team numbers with random rotation step (positive or negative). """
72      # Choose random rotation step
73      step : int = random.randint(1, particle.parameters['teams_num'] - 1)
74      sign: list[int] = random.sample([-1, 1], 1)
75      return rotate_teams(particle, step * sign[0])
```

Πίνακας 8: Συναρτήσεις επιλογής τυχαίων ευρετικών τελεστών

Έτσι, η μετάλλαξη ενός σωματιδίου υλοποιείται όπως φαίνεται στον Πίνακα 9. Να παρατηρήσουμε ότι στις γραμμές 7 – 12 καθορίζεται η πιθανότητα επιλογής των τελεστών σύμφωνα με το βάρος τους (Πίνακας 3).

```python
def mutate_random(particle: Particle) -> Particle:
    """ Generate a new canonical particle with mutation. """
    pm: dict = particle.parameters
    isPhased: bool = True if particle.parameters["gamemode"] == "P" else False
    # Select number of the following heuristic operators
    indeces: list[int] = [1, 2, 3, 4, 5, 6]
    probabilities: list[float] = [pm['operators']['s_h_weight'],
                                  pm['operators']['s_r_weight'],
                                  pm['operators']['s_t_weight'],
                                  pm['operators']['s_p_weight'],
                                  pm['operators']['s_m_weight'],
                                  pm['operators']['r_t_weight']]
    random_selection: list = random.choices(indeces, weights = probabilities, k = 1)
    match random_selection[0]:
        case 1:
            m_particle: Particle = swap_homes_random(particle)
        case 2:
            m_particle: Particle = swap_rounds_random(particle, isPhased)
        case 3:
            m_particle: Particle = swap_teams_random(particle)
        case 4:
            m_particle: Particle = swap_opponents_random(particle)
        case 5:
            m_particle: Particle = swap_matches_random(particle, isPhased)
        case 6:
            m_particle: Particle = rotate_teams_random(particle)
        case _:
            pass
    return m_particle
```

Πίνακας 9: Μετάλλαξη σωματιδίου

Η υλοποίηση της διασταύρωσης και της τυχαίας επιλογής γίνεται με τις παρακάτω συναρτήσεις που φαίνονται στον Πίνακα 10.

```python
def crossover_match(x_particle: Particle, selected_match: tuple, selected_slot: int, isPhased: bool) -> Particle:
    """
    Selected_match located in selected_slot in source particle (like gBest particle)
    is reflected in the same location of x_particle.
    """
    x: list = convert_xis_to_xs(x_particle.position)
    # Find the slot s2 where selected_match is in x_particle
    s2: int = 0
    for s, round in enumerate(x):
        if selected_match in round:
            s2 = s
            break
    # If slots selected_slot, s2 are different, then we move on to crossover.
    # Otherwise, crossover is like it is already done.
    if selected_slot != s2:
        tn: int = x_particle.parameters['slots_num'] // 2
        # If x is phased but s1, s2 do not belong to the same phase,
        if isPhased and ((selected_slot < tn and s2 >= tn) or (selected_slot >= tn and s2 < tn)):
            # then, swap phases of x
            for s in range(tn):
                temp: list = copy.deepcopy(x[s])
                x[s] = copy.deepcopy(x[s + tn])
```

```
                x[s + tn] = temp
            # and correct the position of s2
            if s2 > selected_slot:
                s2 -= tn
            else:
                s2 += tn
        # Finally, swap selected_slot with s2
        temp = copy.deepcopy(x[selected_slot])
        x[selected_slot] = copy.deepcopy(x[s2])
        x[s2] = temp
        z_is = convert_xs_to_xis(x, x_particle.parameters['teams_num'])
        return Particle(z_is, x_particle.parameters)
    return x_particle


def crossover_match_random(x_particle: Particle, y_particle: Particle, isPhased: bool) -> Particle:
    """
    Select a random match_y located at slot_y slot of y.
    Then, swap slots in x_s so that the match_y will end up at slot_y of x.
    """
    y: list = convert_xis_to_xs(y_particle.position)
    # Select a random match of y
    slot_y: int = random.randint(0, y_particle.parameters['slots_num'] - 1)
    match_y: tuple = y[slot_y][random.randint(0, len(y[slot_y]) - 1)]
    # Execute crossover of this match with x
    return crossover_match(x_particle, match_y, slot_y, isPhased)
```

Πίνακας 10: Διασταύρωση σωματιδίου με άλλο σωματίδιο

7.2.2 Πλήρεις τελεστές

Η υλοποίηση των πλήρων τελεστών χρησιμοποιεί τους υπάρχοντες τελεστές διατρέχοντας όλες τις δυνατές επιλογές ορισμάτων τους. Στον Πίνακα 11 φαίνεται η υλοποίησή τους (σύγχρονη και ασύγχρονη).

```
def swap_homes_complete(initial_particle: Particle) -> Particle:
    """
    Tests swap_homes for each (i, j) of a particle.
    Returns the best feasible particle found.
    """
    best_particle: Particle = initial_particle.clone()
    # for each match (i,j)
    for i in tqdm(range(initial_particle.parameters['teams_num']), desc="Progress"):
        for j in range(initial_particle.parameters['teams_num']):
            if i < j:
                m_particle: Particle = swap_homes(initial_particle, i, j)
                if m_particle < best_particle:
                    best_particle = m_particle.clone()
    return best_particle


def swap_rounds_complete(initial_particle: Particle, isPhased: bool) -> Particle:
    """
    Tests swap_rounds for each (s1, s2) of a particle.
    Returns the best particle found.
    """
```

```
    best_particle: Particle = initial_particle.clone()
    phase1: range = range(0, initial_particle.parameters['slots_num'] // 2)
    phase2: range = range(initial_particle.parameters['slots_num'] // 2, initial_particle.parameters['slots_num'])
    # for each slot pair (s1, s2)
    for s1 in tqdm(range(initial_particle.parameters['slots_num']), desc="Progress"):
        for s2 in range(initial_particle.parameters['slots_num']):
            if s1 < s2:
                if isPhased:
                    if ((s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2)):
                        m_particle: Particle = swap_rounds(initial_particle, s1, s2)
                        if m_particle < best_particle:
                            best_particle = m_particle.clone()
                else:
                    m_particle: Particle = swap_rounds(initial_particle, s1, s2)
                    if m_particle < best_particle:
                        best_particle = m_particle.clone()
    return best_particle


def swap_teams_complete(initial_particle: Particle) -> Particle:
    """
    Tests swap_teams for each (i, j) with i < j of a particle.
    Returns the best particle found.
    """
    best_particle: Particle = initial_particle.clone()
    # for each match (i,j)
    for i in tqdm(range(initial_particle.parameters['teams_num']), desc="Progress"):
        for j in range(initial_particle.parameters['teams_num']):
            if i < j:
                m_particle: Particle = swap_teams(initial_particle, i, j)
                if m_particle < best_particle:
                    best_particle = m_particle.clone()
    return best_particle


def swap_opponents_complete(initial_particle: Particle) -> Particle:
    """
    Tests swap_opponents for each (i, j) with i < j of a particle.
    Returns the best particle found.
    """
    best_particle: Particle = initial_particle.clone()
    # for each match (i,j)
    for i in tqdm(range(initial_particle.parameters['teams_num']), desc="Progress"):
        for j in range(initial_particle.parameters['teams_num']):
            if i < j:
                m_particle = swap_opponents(initial_particle, i, j)
                if m_particle < best_particle:
                    best_particle = m_particle.clone()
    return best_particle


def swap_matches_complete(initial_particle: Particle, isPhased: bool) -> Particle:
    """
    Tests swap_matches for each slots pair (s1, s2) and each match of slot s1 of a particle.
    Return the best particle found.
    """
    best_particle: Particle = initial_particle.clone()
    phase1: range = range(0, initial_particle.parameters['slots_num'] // 2)
    phase2: range = range(initial_particle.parameters['slots_num'] // 2, initial_particle.parameters['slots_num'])
    for s1 in tqdm(range(initial_particle.parameters['slots_num']), desc="Progress"):
        for s2 in range(initial_particle.parameters['slots_num']):
            if s1 < s2:
                if isPhased:
                    if (s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2):
                        # In a timetable in x_s format, each row has (teams_num // 2) matches
```

```
                        for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                            m_particle: Particle = swap_matches(initial_particle, s1, s2, match_index_at_slot_s1)
                            if m_particle < best_particle:
                                best_particle = m_particle.clone()
                else:
                    # In a timetable in x_s format, each row has (teams_num // 2) matches
                    for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                        m_particle: Particle = swap_matches(initial_particle, s1, s2, match_index_at_slot_s1)
                        if m_particle < best_particle:
                            best_particle = m_particle.clone()
    return best_particle


def rotate_teams_complete(initial_particle: Particle) -> Particle:
    """
    Rotates teams clockwise and counterclockwise of a particle.
    Returns the best particle found.
    """
    best_particle: Particle = initial_particle.clone()
    for step in tqdm(range(1, initial_particle.parameters['teams_num']), desc="Progress"):
        m_particle: Particle = rotate_teams(initial_particle, step)
        if m_particle < best_particle:
            best_particle = m_particle.clone()
        m_particle: Particle = rotate_teams(initial_particle, -step)
        if m_particle < best_particle:
            best_particle = m_particle.clone()
    return best_particle


def crossover_match_complete(x_particle: Particle, y_particle: Particle, isPhased: bool) -> Particle:
    """ Crossover match of the y particle. Returns the best particle found. """
    best_particle: Particle = x_particle.clone()
    x: list = convert_xis_to_xs(x_particle.position)
    y: list = convert_xis_to_xs(y_particle.position)
    for s, round in enumerate(y):
        for match in round:
            # Take those matches whose location in y is different than in x
            if match not in x[s]:
                m_particle: Particle = crossover_match(x_particle, match, s, isPhased)
                if m_particle < best_particle:
                    best_particle = m_particle.clone()
    return best_particle


def swap_matches_crossover_complete(initial_particle: Particle, isPhased: bool) -> Particle:
    """
    Swap_matches for each slots pair (s1, s2) and for each match of slot s1 of a particle.
    Then, crossover complete the resulted particle with the initial particle and returns
    the best particle found.
    """
    best_particle: Particle = initial_particle.clone()
    phase1: range = range(0, initial_particle.parameters['slots_num'] // 2)
    phase2: range = range(initial_particle.parameters['slots_num'] // 2, initial_particle.parameters['slots_num'])
    for s1 in tqdm(range(initial_particle.parameters['slots_num']), desc="Progress"):
        for s2 in range(initial_particle.parameters['slots_num']):
            if s1 < s2:
                if isPhased:
                    if (s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2):
                        # In a timetable in x_s format, each row has (teams_num // 2) matches
                        for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                            s_particle: Particle = swap_matches(initial_particle, s1, s2, match_index_at_slot_s1)
                            m_particle: Particle = crossover_match_complete(s_particle,initial_particle, isPhased)
                            if m_particle < best_particle:
                                best_particle = m_particle.clone()
                else:
```

```
                    # In a timetable in x_s format, each row has (teams_num // 2) matches
                    for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                        s_particle: Particle = swap_matches(initial_particle, s1, s2, match_index_at_slot_s1)
                        m_particle: Particle = crossover_match_complete(s_particle, initial_particle, isPhased)
                        if m_particle < best_particle:
                            best_particle = m_particle.clone()
        return best_particle

    def unpack_args_and_swap(args: tuple) -> Particle:
        """ Unpacks the arguments and calls swap_homes. """
        particle: Particle
        i: int
        j: int
        particle, i, j = args
        return swap_homes(particle, i, j)

    def swap_homes_complete_async(initial_particle: Particle) -> Particle:
        """
        Tests swap_homes for each (i, j) of a particle using multiprocessing.
        Returns the best particle found.
        """
        best_particle: Particle = initial_particle.clone()

        # Prepare a list of (i, j) pairs to be tested in parallel
        tasks = [(initial_particle, i, j) for i in range(initial_particle.parameters['teams_num'])
                                          for j in range(i + 1, initial_particle.parameters['teams_num'])]
        # Execute swaps in parallel using processes
        with concurrent.futures.ProcessPoolExecutor() as executor:
            results = list(tqdm(executor.map(unpack_args_and_swap, tasks), desc="Swap homes", total=len(tasks)))
        # Evaluate the results
        for result in results:
            if result < best_particle:
                best_particle = result.clone()
        return best_particle


    def unpack_args_and_swap_rounds(args: tuple) -> Particle:
        """ Unpack the arguments and call swap_rounds. """
        particle: Particle
        s1: int
        s2: int
        particle, s1, s2 = args
        return swap_rounds(particle, s1, s2)

    def swap_rounds_complete_async(initial_particle: Particle, isPhased: bool) -> Particle:
        """
        Tests swap_rounds for each slots pair (s1, s2) of a particle using multiprocessing.
        Return the best particle found.
        """
        best_particle: Particle = initial_particle.clone()
        slots_num = initial_particle.parameters['slots_num']
        phase1 = range(0, slots_num // 2)
        phase2 = range(slots_num // 2, slots_num)
        # Prepare a list of (s1, s2) pairs to be tested in parallel
        if isPhased:
            tasks = [(initial_particle, s1, s2) for s1 in range(slots_num) for s2 in range(slots_num)
                        if s1 < s2 and ((s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2))]
        else:
            tasks = [(initial_particle, s1, s2) for s1 in range(slots_num) for s2 in range(slots_num) if s1 < s2]
        # Execute swaps in parallel using processes
        with concurrent.futures.ProcessPoolExecutor() as executor:
            results = list(tqdm(executor.map(unpack_args_and_swap_rounds, tasks),
                                                                    desc="Swap rounds", total=len(tasks)))
        # Evaluate the results
        for result in results:
```

```
215             if result < best_particle:
216                 best_particle = result.clone()
217         return best_particle
218 
219 
220     def unpack_args_and_swap_teams(args: tuple) -> Particle:
221         """ Unpack the arguments and call swap_teams. """
222         particle: Particle
223         i: int
224         j: int
225         particle, i, j = args
226         return swap_teams(particle, i, j)
227 
228     def swap_teams_complete_async(initial_particle: Particle) -> Particle:
229         """
230         Tests swap_teams for each pair (i, j) with i < j of a particle using multiprocessing.
231         Return the best particle found.
232         """
233         best_particle: Particle = initial_particle.clone()
234         # Prepare a list of (i, j) pairs to be tested in parallel
235         tasks = [(initial_particle, i, j) for i in range(initial_particle.parameters['teams_num'])
236                                           for j in range(i + 1, initial_particle.parameters['teams_num'])]
237         # Execute swaps in parallel using processes
238         with concurrent.futures.ProcessPoolExecutor() as executor:
239             results = list(tqdm(executor.map(unpack_args_and_swap_teams, tasks), desc="Swap teams", total=len(tasks)))
240         # Evaluate the results
241         for result in results:
242             if result < best_particle:
243                 best_particle = result.clone()
244         return best_particle
245 
246 
247     def unpack_args_and_swap_opponents(args: tuple) -> Particle:
248         """ Unpack the arguments and call swap_opponents. """
249         particle: Particle
250         i: int
251         j: int
252         particle, i, j = args
253         return swap_opponents(particle, i, j)
254 
255     def swap_opponents_complete_async(initial_particle: Particle) -> Particle:
256         """
257         Tests swap_opponents for each pair (i, j) with i < j of a particle using multiprocessing.
258         Returns the best particle found.
259         """
260         best_particle: Particle = initial_particle.clone()
261         # Prepare a list of (i, j) pairs to be tested in parallel
262         tasks = [(initial_particle, i, j) for i in range(initial_particle.parameters['teams_num'])
263                                           for j in range(i + 1, initial_particle.parameters['teams_num'])]
264         # Execute swaps in parallel using processes
265         with concurrent.futures.ProcessPoolExecutor() as executor:
266             results = list(tqdm(executor.map(unpack_args_and_swap_opponents, tasks), desc="Swap opponents",
267                                                                                      total=len(tasks)))
268         # Evaluate the results
269         for result in results:
270             if result < best_particle:
271                 best_particle = result.clone()
272         return best_particle
273 
274 
275     def unpack_args_and_swap_matches(args: tuple) -> Particle:
276         """ Unpack the arguments and call swap_matches. """
277         particle: Particle
278         s1: int
279         s2: int
```

```
        match_index_at_slot_s1: int
        particle, s1, s2, match_index_at_slot_s1 = args
        return swap_matches(particle, s1, s2, match_index_at_slot_s1)

    def swap_matches_complete_async(initial_particle: Particle, isPhased: bool) -> Particle:
        """
        Tests swap_matches for each (s1, s2) and for each match of
        slot s1 of a particle using multiprocessing.
        Returns the best particle found.
        """
        best_particle: Particle = initial_particle.clone()
        slots_num = initial_particle.parameters['slots_num']
        phase1 = range(0, slots_num // 2)
        phase2 = range(slots_num // 2, slots_num)
        tasks = []
        # Collect tasks considering whether the scheduling is phased or not
        for s1 in range(slots_num):
            for s2 in range(slots_num):
                if s1 < s2:
                    if isPhased:
                        if (s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2):
                            # Prepare tasks for each match in the slot
                            for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                                tasks.append((initial_particle, s1, s2, match_index_at_slot_s1))
                    else:
                        # Prepare tasks for each match in the slot
                        for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                            tasks.append((initial_particle, s1, s2, match_index_at_slot_s1))
        # Execute swaps in parallel using processes
        with concurrent.futures.ProcessPoolExecutor() as executor:
            results = list(tqdm(executor.map(unpack_args_and_swap_matches, tasks), desc="Swap matches",
                                                                                   total=len(tasks)))
        # Evaluate the results
        for result in results:
            if result < best_particle:
                best_particle = result.clone()
        return best_particle


    def unpack_args_and_rotate_teams(args: tuple) -> Particle:
        """ Unpack the arguments and call rotate_teams. """
        particle: Particle
        step: int
        particle, step = args
        return rotate_teams(particle, step)

    def rotate_teams_complete_async(initial_particle: Particle) -> Particle:
        """
        Rotates teams to the right and to the left of a particle using multiprocessing.
        Returns the best particle found.
        """
        best_particle: Particle = initial_particle.clone()
        # Prepare a list of steps to be tested in parallel
        tasks = [(initial_particle, step) for step in range(1, initial_particle.parameters['teams_num'])] + \
                [(initial_particle, -step) for step in range(1, initial_particle.parameters['teams_num'])]
        # Execute rotations in parallel using processes
        with concurrent.futures.ProcessPoolExecutor() as executor:
            results = list(tqdm(executor.map(unpack_args_and_rotate_teams, tasks), desc="Rotate teams",
                                                                                   total=len(tasks)))
        # Evaluate the results
        for result in results:
            if result < best_particle:
                best_particle = result.clone()
        return best_particle
```

```python
def unpack_args_and_crossover(args: tuple) -> Particle:
    """ Unpack the arguments and call crossover_match. """
    x_particle: Particle
    match: tuple
    slot: int
    isPhased: bool
    x_particle, match, slot, isPhased = args
    return crossover_match(x_particle, match, slot, isPhased)

def crossover_match_complete_async(x_particle: Particle, y_particle: Particle, isPhased: bool) -> Particle:
    """
    Crossover each match of the y particle using multiprocessing.
    Return the best particle found.
    """
    best_particle: Particle = x_particle.clone()
    x = convert_xis_to_xs(x_particle.position)
    y = convert_xis_to_xs(y_particle.position)
    # Prepare a list of tasks to be processed in parallel
    tasks = []
    for s, round in enumerate(y):
        for match in round:
            if match not in x[s]:
                tasks.append((x_particle, match, s, isPhased))
    # Execute crossover matches in parallel using processes
    with concurrent.futures.ProcessPoolExecutor() as executor:
        results = executor.map(unpack_args_and_crossover, tasks)
    # Evaluate the results
    for result in results:
        if result < best_particle:
            best_particle = result.clone()
    return best_particle


def unpack_args_and_swap_and_crossover(args: tuple) -> Particle:
    """ Unpack the arguments and call swap_and_crossover. """
    x_particle: Particle
    s1: int
    s2: int
    match_index_at_slot_s1: int
    isPhased: bool
    x_particle, s1, s2, match_index_at_slot_s1, isPhased = args
    s_particle = swap_matches(x_particle, s1, s2, match_index_at_slot_s1)
    m_particle = crossover_match_complete(s_particle, x_particle, isPhased)
    if m_particle < x_particle:
        m_particle.save()
        m_particle.print()
    return m_particle

def swap_matches_crossover_complete_async(initial_particle: Particle, isPhased: bool) -> Particle:
    """
    Swap_matches for each (s1, s2) and for each match of slot s1 of a particle.
    Then, crossover complete the resulted particle with the initial particle and return
    the best particle found using multiprocessing.
    """
    best_particle: Particle = initial_particle.clone()
    slots_num = initial_particle.parameters['slots_num']
    phase1 = range(0, slots_num // 2)
    phase2 = range(slots_num // 2, slots_num)
    tasks = []
    # Collect tasks considering whether the scheduling is phased or not
    for s1 in range(slots_num):
        for s2 in range(slots_num):
            if s1 < s2:
                if isPhased:
                    if ((s1 in phase1 and s2 in phase1) or (s1 in phase2 and s2 in phase2)):
```

```
                        # Prepare tasks for each match in the slot
                        for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                            tasks.append((initial_particle, s1, s2, match_index_at_slot_s1, isPhased))
                else:
                    # Prepare tasks for each match in the slot
                    for match_index_at_slot_s1 in range(initial_particle.parameters['teams_num'] // 2):
                        tasks.append((initial_particle, s1, s2, match_index_at_slot_s1, False))
    # Execute swap and crossover in parallel using processes
    with concurrent.futures.ProcessPoolExecutor() as executor:
        results = list(tqdm(executor.map(unpack_args_and_swap_and_crossover, tasks),
                                                    desc="Swap matches crossover", total=len(tasks)))
    # Evaluate the results
    for result in results:
        if result < best_particle:
            best_particle = result.clone()
    return best_particle
```

Πίνακας 11: Πλήρεις τελεστές (σύγχρονοι και ασύγχρονοι)

7.2.3 Βοηθητικές συναρτήσεις CP-SAT

Για την υλοποίηση του αλγορίθμου CP-SAT Enhancer, θα χρειαστούμε αρχικώς δύο βοηθητικές συναρτήσεις: την συνάρτηση *fix_entities* και την συνάρτηση *evaluate_breaks* (Πίνακας 12) . Η πρώτη συνάρτηση δέχεται ως όρισμα ένα ωρολόγιο πρόγραμμα (σε κωδικοποίηση $g_{i,j,s}$), το πλήθος των οντοτήτων (ομάδων ή χρονοθυρίδων) που επιθυμούμε να κρατήσουμε σταθερές (*fix_num*) και το είδος της οντότητας (*mode*). Έτσι, αν για παράδειγμα ζητήσουμε να κρατήσουμε σταθερές 5 ομάδες, η συνάρτηση *fix_entities* επιλέγει τυχαία 5 ομάδες από το σύνολο των ομάδων και επιστρέφει μία λίστα που περιλαμβάνει όλα εκείνα τα παιχνίδια στα οποία συμμετέχουν αυτές οι 5 ομάδες.

Η συνάρτηση *evaluate_breaks* δέχεται ως όρισμα ένα ωρολόγιο πρόγραμμα (σε κωδικοποίηση $g_{i,j,s}$), την ομάδα που μας ενδιαφέρει και μία λίστα από χρονοθυρίδες και επιστρέφει το πλήθος των *breaks* που έχει αυτή η ομάδα στο σύνολο των δοσμένων χρονοθυρίδων.

```
def fix_entities(g: np.ndarray, fix_num: int, parameters: dict, mode: str) -> list:
    """
    Filters the games that have been played among a randomly selected
    subset M of entities (teams or slots).

    Arguments:
    - g: A 3D array indicating whether a game [i,j,s] has been played.
    - fix_num: The number of entities (teams or slots) to be fixed.
    - parameters: The parameters of the problem.
    - mode: The mode of operation ('team' or 'slot') specifying whether to fix teams or slots.
```

```
    Returns:
    - A list of tuples (i, j, s) for games that have been played among entities in M.
    """
    entities_num: int = parameters["teams_num"] if mode == 'team' else parameters["slots_num"]
    if fix_num > entities_num:
        fix_num = entities_num
    if fix_num < 1:
        fix_num = 0

    # Select the fixed entities randomly
    M: list = random.sample(range(entities_num), fix_num)
    played_games: list = []
    if mode == 'team':
        for i in M:
            for j in M:
                if i != j:  # Avoid a team playing against itself
                    for s in range(parameters["slots_num"]):
                        if g[i, j, s] == 1:
                            played_games.append((i, j, s))
    else:  # mode == 'slot'
        for s in M:
            for i in range(parameters["teams_num"]):
                for j in range(parameters["teams_num"]):
                    if i != j:  # Avoid a team playing against itself
                        if g[i, j, s] == 1:
                            played_games.append((i, j, s))
    return played_games


def evaluate_breaks(model, g, team: int, slots: list, parameters: dict) -> int:
    """
    Evaluate and count the number of 'breaks' for a given team across specified slots.
    A 'break' is defined as having two consecutive games either both at home or both away.

    Arguments:
    - model: The CP-SAT model instance.
    - g: Dictionary of game variables, g[(i, j, s)] = 1 if team i plays at home against team j in slot s.
    - team: The team for which to evaluate breaks.
    - slots: Ordered list of slots to consider for evaluating breaks.
    - parameters: The parameters of the problem.

    Returns:
    - The total number of breaks for the team.
    """
    break_vars: list = []
    teams_num: int = parameters["teams_num"]
    # For each slot s in slots
    for s in slots:
        if s > 0:  # Check for slots > 0. Slot 0 cannot have breaks.
            s_prev = s - 1
            break_var = model.NewBoolVar(f'break_{team}_{s}')
            home_game_current = model.NewIntVar(0, 1, f'home_game_current_{team}_{s}')
            home_game_prev = model.NewIntVar(0, 1, f'home_game_prev_{team}_{s}')

            # Determine if team is playing a home or away game at slot s.
            # Take the sum of homes games of team with all j at s. It must be 1 (if team plays at home)
            # or 0 (if not). Same for the slot s - 1.
            model.Add(home_game_current == sum([g[(team, j, s)] for j in range(teams_num) if j != team]))
            model.Add(home_game_prev == sum([g[(team, j, s_prev)] for j in range(teams_num) if j != team]))

            # Break condition: same status in current and previous games
            # If team plays a home game at slot s and s - 1, then it has a break.
            # If team doesn't play a home game neither at slot s not at slot s - 1, then it plays away at
            # both slots and then, it has a break. Otherwise, there is not break at slot s.
            model.Add(break_var == 1).OnlyEnforceIf([home_game_current, home_game_prev])

```

```
            model.Add(break_var == 1).OnlyEnforceIf([home_game_current.Not(), home_game_prev.Not()])
            # Gather all break variables
            break_vars.append(break_var)

    # Return the total number of break the team has in slots
    return sum(break_vars)
```

Πίνακας 12: Βοηθητικές συναρτήσεις CP-SAT

Οι επόμενες βοηθητικές συναρτήσεις που χρειαζόμαστε είναι οι:

- *add_structural_constraints*
- *add_hard_constraints*
- *add_constraints_as_soft*
- *cpsat_task*

```
def add_structural_constraints(model, g, teams_num, slots_num, isPhased) -> None:
    """
    Adds structural constraints to the CP-SAT model for generating a sports timetable.

    Arguments:
    - model: The CP-SAT model.
    - g: A dictionary of decision variables, where g[i, j, s] == 1
         if team i plays at home against team j in slot s.
    - teams_num: The number of teams.
    - slots_num: The total number of slots in the schedule.
    - isPhased: True if the timetable is phased.
    Returns:
    - None
    """
    # Rule 0: Each team plays exactly once per slot.
    # Rule 1 is satisfied by the model variables declaration.
    for s in range(slots_num):
        for i in range(teams_num):
            # Sum over all j != i, g[i, j, s] + g[j, i, s] == 1
            model.Add(sum(g[(i, j, s)] + g[(j, i, s)] for j in range(teams_num) if i != j) == 1)
    for i in range(teams_num):
        for j in range(i + 1, teams_num):
            home_games_ij: list = [g[(i, j, s)] for s in range(slots_num)]
            away_games_ij: list = [g[(j, i, s)] for s in range(slots_num)]

            # Round Robin Rule: Each team plays twice with each other team: once at home and once away.
            model.Add(sum(home_games_ij) == 1)
            model.Add(sum(away_games_ij) == 1)

            # Phase Rule: Each team plays once with every other team in each phase. Assuming RR = 2.
            # Condition for phase 2 is implied by round-robin rule.
            if isPhased:
                phase_length: int = slots_num // 2
                model.Add(sum(home_games_ij[:phase_length]) + sum(away_games_ij[:phase_length]) == 1)
```

Πίνακας 13: CP-SAT: συνάρτηση add_structural_constraints

Η συνάρτηση *add_structural_constraints* προσθέτει στο μοντέλο τους δομικούς περιορισμούς του προβλήματος. Οι περιορισμοί αυτοί, ως γνωστόν, πρέπει να ικανοποιούνται πάντα. Η υλοποίηση της συνάρτησης αυτής φαίνεται στον Πίνακα 13.

Η συνάρτηση *add_hard_constraints* προσθέτει τους ισχυρούς περιορισμούς του προβλήματος στο μοντέλο CP-SAT και υλοποιείται σε γλώσσα Python όπως φαίνεται στον ακόλουθο πίνακα:

```python
def add_hard_constraints(model, g, parameters: dict) -> None:
    """
    Adds the hard constraints into the model.

    Arguments:
    - model: The CP-SAT model.
    - g: The game variables dictionary, g[(i, j, s)] == 1
         if team i plays at home against team j at slot s.
    - parameters: The dictionary containing the problem parameters.

    Returns:
    - None
    """
    teams_num: int = parameters["teams_num"]
    slots_num: int = parameters["slots_num"]
    constraints: dict = parameters["constraints"]
    constraint_category_found: bool

    # Create a list of all ctype constraint keys
    keys: list = [key for key in constraints if constraints[key]["ctype"] == "HARD"]

    for key in keys:
        constraint_category_found = False

        # CA1 constraint
        if constraints[key]["category"] == "CA1":
            constraint_category_found = True
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            slots: list = constraints[key]["timeslots"]
            for i in constraints[key]["teams1"]:
                if constraints[key]["mode1"] == "H":
                    home_games: int = sum(g[i, j, s] for j in range(teams_num) if i != j for s in slots)
                    # Sum home games for team i in the given slots and enforce the min and max constraints
                    if k_min > 0:
                        model.Add(k_min <= home_games)
                    model.Add(home_games <= k_max)
                else:
                    away_games: int = sum(g[j, i, s] for j in range(teams_num) if i != j for s in slots)
                    # Sum away games for team i in the given slots and enforce the min and max constraints
                    if k_min > 0:
                        model.Add(k_min <= away_games)
                    model.Add(away_games <= k_max)

        # CA2 constraint
        if constraints[key]["category"] == "CA2":
            constraint_category_found = True
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
```

```
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            teams2: list = constraints[key]["teams2"]
            for i in teams1:
                if constraints[key]["mode1"] == 'H':
                    home_games: int = sum(g[i, j, s] for j in teams2 if i != j for s in slots)
                    # Count all home games team i plays against teams in T2 across specified slots
                    if k_min > 0:
                        model.Add(k_min <= home_games)
                    model.Add(home_games <= k_max)
                elif constraints[key]["mode1"] == 'A':
                    away_games: int = sum(g[j, i, s] for j in teams2 if i != j for s in slots)
                    # Count all away games team i plays against teams in T2 across specified slots
                    if k_min > 0:
                        model.Add(k_min <= away_games)
                    model.Add(away_games <= k_max)
                else:  # 'HA'
                    # Count all games (home and away) team i plays against teams in T2 across specified slots
                    total_games: int = sum(g[i, j, s] for j in teams2 if i != j for s in slots) + \
                                sum(g[j, i, s] for j in teams2 if i != j for s in slots)
                    if k_min > 0:
                        model.Add(k_min <= total_games)
                    model.Add(total_games <= k_max)

        # CA3 constraint (mode2 = SLOTS only)
        if constraints[key]["category"] == "CA3":
            constraint_category_found = True
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            k : int = constraints[key]["k"]
            teams1: list = constraints[key]["teams1"]
            teams2: list = constraints[key]["teams2"]
            # Apply CA3 constraint for each team in teams1 against
            # teams in teams2 in each sliding window of k slots
            for i in teams1:
                for start_slot in range(slots_num - k + 1):
                    count = model.NewIntVar(0, k_max, f'count_{i}_{start_slot}')
                    match constraints[key]["mode1"]:
                        case 'H':
                            model.Add(count == sum(g[(i, j, s)] for j in teams2
                                                   for s in range(start_slot, start_slot + k) if i != j))
                        case 'A':
                            model.Add(count == sum(g[(j, i, s)] for j in teams2
                                                   for s in range(start_slot, start_slot + k) if i != j))
                        case 'HA':
                            model.Add(count == sum(g[(i, j, s)] + g[(j, i, s)] for j in teams2
                                                   for s in range(start_slot, start_slot + k) if i != j))
                    model.Add(count >= k_min)
                    model.Add(count <= k_max)

        # CA4 constraint
        if constraints[key]["category"] == "CA4":
            constraint_category_found = True
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            teams2: list = constraints[key]["teams2"]
            mode1: str = constraints[key]["mode1"]
            mode2: str = constraints[key]["mode2"]
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            match mode2:
                case "GLOBAL":
                    match mode1:
                        case "H":
                            sum_g: int = sum(g[(t1, t2, s)] for t1 in teams1 for t2 in teams2
```

```
                                                                    for s in slots if t1 != t2)
                        case "A":
                            sum_g: int = sum(g[(t2, t1, s)] for t1 in teams1 for t2 in teams2
                                                                    for s in slots if t1 != t2)
                        case "HA":
                            sum_g: int = sum(g[(t1, t2, s)] + g[(t1, t2, s)] for t1 in teams1
                                                                for t2 in teams2 for s in slots if t1 != t2)
                    model.Add(sum_g <= k_max)
                    if k_min > 0:
                        model.Add(sum_g >= k_min)
                case "EVERY":
                    for s in slots:
                        match mode1:
                            case "H":
                                sum_every:int = sum(g[(t1, t2, s)] for t1 in teams1 for t2 in teams2 if t1 != t2)
                            case "A":
                                sum_every:int = sum(g[(t2, t1, s)] for t1 in teams1 for t2 in teams2 if t1 != t2)
                            case "HA":
                                sum_every: int = sum(g[(t1, t2, s)] + g[(t2, t1, s)] for t1 in teams1
                                                                    for t2 in teams2 if t1 != t2)
                        model.Add(sum_every <= k_max)
                        if k_min > 0:
                            model.Add(sum_every >= k_min)

        # GA1 constraint
        if constraints[key]["category"] == "GA1":
            constraint_category_found = True
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            slots: list = constraints[key]["timeslots"]
            meetings: list = constraints[key]["meetings"]
            # Count the number of games in Q that are scheduled within the slots S
            games_in_slots: int = sum(g[i, j, s] for (i, j) in meetings for s in slots)
            if k_min > 0:
                model.Add(k_min <= games_in_slots)
            model.Add(games_in_slots <= k_max)

        # BR1 constraint (HA only)
        if constraints[key]["category"] == "BR1":
            constraint_category_found = True
            teams1: list = constraints[key]["teams1"]
            slots: list = constraints[key]["timeslots"]
            k_max: int = constraints[key]["k"]
            if constraints[key]["mode2"] == "HA":
                for i in teams1:
                    # For each team in teams1, count the number of breaks.
                    breaks = []
                    breaks.append(evaluate_breaks(model, g, i, slots, parameters))
                    # Add constraint on the total number of breaks
                    if constraints[key]["mode1"] == "LEQ":
                        model.Add(sum(breaks) <= k_max)
                    else:
                        model.Add(sum(breaks) == k_max)

        # BR2 constraint (HA only)
        if constraints[key]["category"] == "BR2":
            constraint_category_found = True
            teams1: list = constraints[key]["teams1"]
            slots: list = constraints[key]["timeslots"]
            k_max: int = constraints[key]["k"]
            if constraints[key]["mode1"] == "HA":
                break_vars_for_teams: list = [evaluate_breaks(model, g, i, slots, parameters) for i in teams1]
                total_breaks = model.NewIntVar(0, sum(slots), 'total_breaks_aggregate')
                model.Add(sum(break_vars_for_teams) == total_breaks)
                if constraints[key]["mode2"] == "LEQ":
```

```
                    model.Add(total_breaks <= k_max)
                else:
                    model.Add(total_breaks == k_max)

        # FA2 constraint
        if constraints[key]["category"] == "FA2":
            constraint_category_found = True
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            mode2: str = constraints[key]["mode2"]
            k: int = constraints[key]["k"]
            if mode2 == "H":
                for i in teams1:
                    for j in teams1:
                        if i < j:
                            a: list = [0]
                            for s in slots:
                                abs_diff2 = model.NewIntVar(0, 100, 'abs_diff2_{s}')
                                model.AddAbsEquality(abs_diff2, sum(g[(i, t, p)] for t in teams1
                                        for p in range(s + 1) if i != t) -
                                        sum(g[(j, t, p)] for t in teams1 for p in range(s + 1) if j != t))
                                model.Add(abs_diff2 - k <= 0)

        # SE1 constraint
        if constraints[key]["category"] == "SE1":
            constraint_category_found = True
            teams1: list = constraints[key]["teams1"]
            k_min: int = constraints[key]["k_min"]
            for i in teams1:
                for j in teams1:
                    if i < j:
                        abs_d = model.NewIntVar(0, slots_num, f"abs_d_{i}_{j}")
                        model.AddAbsEquality(abs_d, sum((g[(i,j,s)] - g[(j,i,s)]) * s for s in range(slots_num)))
                        model.Add(abs_d - 1 >= k_min)

        if constraint_category_found == False:
            print(f"Error! Hard constraint {constraints[key]['category']} not set up.")
```

Πίνακας 14: CP-SAT: συνάρτηση add_hard_constraints

Η τρίτη συνάρτηση, *add_constraints_as_soft* προσθέτει τους ασθενείς περιορισμούς του προβλήματος στο μοντέλο CP-SAT, αλλά μπορεί να χρησιμοποιηθεί για να προσθέσει τους ισχυρούς περιορισμούς του προβλήματος ως ασθενείς επίσης. Η επιλογή της λειτουργίας αυτής γίνεται με την βοήθεια της μεταβλητής *ctype*: αν λάβει τιμή *HARD*, τότε ο ισχυρός περιορισμός εισάγεται ως ασθενής στο μοντέλος. Αν λάβει τιμή *SOFT*, τότε ο περιορισμός είναι ασθενής και αντιμετωπίζεται αναλόγως. Η υλοποίηση της συνάρτησης φαίνεται στον Πίνακα 15.

```
def add_constraints_as_soft(model, g, ctype: str, parameters: dict) -> list:
    """
    Adds the hard or soft constraints into the model with penalties for minimization.

    Arguments:
    - model: The CP-SAT model.
```

```
        - g: The game variables dictionary, g[(i, j, s)] == 1 if team i plays at home against team j at slot s.
        - ctype: The type ("HARD" or "SOFT") of the constraint
        - parameters: The dictionary containing the problem parameters.
        Returns:
        - A list of penalties
        """
        penalties: list = []
        teams_num = parameters["teams_num"]
        slots_num = parameters["slots_num"]
        constraints: dict = parameters["constraints"]
        constraint_category_found: bool

        # Create a list of all ctype constraint keys
        keys: list = [key for key in constraints if constraints[key]["ctype"] == ctype]

        for key in keys:
            penalty_weight: int = constraints[key]["penalty"]
            constraint_category_found = False

            # CA1 constraint
            if constraints[key]["category"] == "CA1":
                constraint_category_found = True
                slots: list = constraints[key]["timeslots"]
                teams1: list = constraints[key]["teams1"]
                mode1: str = constraints[key]["mode1"]
                k_min: int = constraints[key]["k_min"]
                k_max: int = constraints[key]["k_max"]
                deviation = model.NewIntVar(0, 10, "deviation")
                for i in teams1:
                    match mode1:
                        case "H":
                            sum_h: int  = sum(g[(i, j, s)] for j in range(teams_num) for s in slots if i != j)
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_h, sum_h - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_h - k_max, 0])
                        case "A":
                            sum_a: int = sum(g[(j, i, s)] for j in range(teams_num) for s in slots if i != j)
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_a, sum_a - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_a - k_max, 0])
                    penalties.append(deviation * penalty_weight)

            # CA2 constraint
            if constraints[key]["category"] == "CA2":
                constraint_category_found = True
                slots: list = constraints[key]["timeslots"]
                teams1: list = constraints[key]["teams1"]
                teams2: list = constraints[key]["teams2"]
                mode1: str = constraints[key]["mode1"]
                k_min: int = constraints[key]["k_min"]
                k_max: int = constraints[key]["k_max"]
                deviation = model.NewIntVar(0, 100, "deviation")
                for t1 in teams1:
                    match mode1:
                        case "H":
                            sum_h: int = sum(g[(t1, t2, s)] for t2 in teams2 for s in slots if t1 != t2)
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_h, sum_h - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_h - k_max, 0])
                        case "A":
                            sum_a: int = sum(g[(t2, t1, s)] for t2 in teams2 for s in slots if t1 != t2)
                            if k_min > 0:
```

```
                            model.AddMaxEquality(deviation, [k_min - sum_a, sum_a - k_max, 0])
                        else:
                            model.AddMaxEquality(deviation, [sum_a - k_max, 0])
                    case "HA":
                        sum_ha:int=sum(g[(t1, t2, s)]+g[(t2, t1, s)] for t2 in teams2 for s in slots if t1 != t2)
                        if k_min > 0:
                            model.AddMaxEquality(deviation, [k_min - sum_ha, sum_ha - k_max, 0])
                        else:
                            model.AddMaxEquality(deviation, [sum_ha - k_max, 0])
                penalties.append(deviation * penalty_weight)

        # CA3 constraint (mode2 = SLOTS only)
        if constraints[key]["category"] == "CA3":
            constraint_category_found = True
            teams1: list = constraints[key]["teams1"]
            teams2: list = constraints[key]["teams2"]
            mode1: str = constraints[key]["mode1"]
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            k: int = constraints[key]["k"]
            for i in teams1:
                for start_slot in range(slots_num - k + 1):
                    count = model.NewIntVar(0, 100, f'count_{i}_{start_slot}')
                    deviation = model.NewIntVar(0, 100, f'deviation_{i}_{start_slot}')
                    excess = model.NewIntVar(0, 100, f'excess_{i}_{start_slot}')
                    deficit = model.NewIntVar(0, 100, f'deficit_{i}_{start_slot}')
                    match mode1:
                        case "H":
                            model.Add(count == sum(g[(i, j, s)] for j in teams2
                                                   for s in range(start_slot, start_slot + k) if i != j))
                        case "A":
                            model.Add(count == sum(g[(j, i, s)] for j in teams2
                                                   for s in range(start_slot, start_slot + k) if i != j))
                        case "HA":
                            model.Add(count == sum(g[(i, j, s)] + g[(j, i, s)]
                                                   for j in teams2 for s in range(start_slot, start_slot + k) if i != j))
                    model.AddMaxEquality(excess, [count - k_max, 0])
                    model.AddMaxEquality(deficit, [k_min - count, 0])
                    model.AddMaxEquality(deviation, [excess, deficit, 0])
                    penalties.append(deviation * penalty_weight)

        # CA4 constraint
        if constraints[key]["category"] == "CA4":
            constraint_category_found = True
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            teams2: list = constraints[key]["teams2"]
            mode1: str = constraints[key]["mode1"]
            mode2: str = constraints[key]["mode2"]
            k_min: int = constraints[key]["k_min"]
            k_max: int = constraints[key]["k_max"]
            match mode2:
                case "GLOBAL":
                    deviation = model.NewIntVar(0, 4000, "deviation")
                    match mode1:
                        case "H":
                            sum_h: int = sum(g[(t1, t2, s)] for t1 in teams1 for t2 in teams2
                                                                    for s in slots if t1 != t2)
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_h, sum_h - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_h - k_max, 0])
                        case "A":
                            sum_a: int = sum(g[(t2, t1, s)] for t1 in teams1 for t2 in teams2
                                                                    for s in slots if t1 != t2)
```

```
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_a, sum_a - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_a - k_max, 0])
                        case "HA":
                            sum_ha: int = sum(g[(t1, t2, s)] + g[(t2, t1, s)] for t1 in teams1
                                                                    for t2 in teams2 for s in slots if t1 != t2)
                            if k_min > 0:
                                model.AddMaxEquality(deviation, [k_min - sum_ha, sum_ha - k_max, 0])
                            else:
                                model.AddMaxEquality(deviation, [sum_ha - k_max, 0])

                    penalties.append(deviation * penalty_weight)
                case "EVERY":
                    for s in slots:
                        deviation = model.NewIntVar(0, 1000, "deviation")
                        match mode1:
                            case "H":
                                sum_h: int = sum(g[(t1, t2, s)] for t1 in teams1 for t2 in teams2 if t1 != t2)
                                if k_min > 0:
                                    model.AddMaxEquality(deviation, [k_min - sum_h, sum_h - k_max, 0])
                                else:
                                    model.AddMaxEquality(deviation, [sum_h - k_max, 0])
                            case "A":
                                sum_a: int = sum(g[(t2, t1, s)] for t1 in teams1 for t2 in teams2 if t1 != t2)
                                if k_min > 0:
                                    model.AddMaxEquality(deviation, [k_min - sum_a, sum_a - k_max, 0])
                                else:
                                    model.AddMaxEquality(deviation, [sum_a - k_max, 0])
                            case "HA":
                                sum_ha: int = sum(g[(t1, t2, s)] + g[(t2, t1, s)] for t1 in teams1
                                                                        for t2 in teams2 if t1 != t2)
                                if k_min > 0:
                                    model.AddMaxEquality(deviation, [k_min - sum_ha, sum_ha - k_max, 0])
                                else:
                                    model.AddMaxEquality(deviation, [sum_ha - k_max, 0])
                        penalties.append(deviation * penalty_weight)

            # GA1 constraint
            if constraints[key]["category"] == "GA1":
                constraint_category_found = True
                slots: list = constraints[key]["timeslots"]
                meetings: list = constraints[key]["meetings"]
                k_min: int = constraints[key]["k_min"]
                k_max: int = constraints[key]["k_max"]
                deviation = model.NewIntVar(0, 100, "deviation")
                sm: int = sum(g[(i, j, s)] for (i, j) in meetings for s in slots)
                if k_min > 0:
                    model.AddMaxEquality(deviation, [k_min - sm, sm - k_max, 0])
                else:
                    model.AddMaxEquality(deviation, [sm - k_max, 0])
                penalties.append(deviation * penalty_weight)

            # BR1 constraint
            if constraints[key]["category"] == "BR1":
                constraint_category_found = True
                slots: list = constraints[key]["timeslots"]
                teams1: list = constraints[key]["teams1"]
                mode1: str = constraints[key]["mode1"]
                mode2: str = constraints[key]["mode2"]
                k: int = constraints[key]["k"]
                if mode2 == "HA":
                    deviation = model.NewIntVar(0, 100, "deviation")
                    total_breaks_for_team: int = evaluate_breaks(model, g, teams1[0], slots, parameters)
                    if mode1 == "LEQ":
```

```
                    model.AddMaxEquality(deviation, [total_breaks_for_team - k, 0])
                else:
                    model.AddMaxEquality(deviation, [total_breaks_for_team - k, k - total_breaks_for_team, 0])
                penalties.append(deviation * penalty_weight)

        # BR2 constraint
        if constraints[key]["category"] == "BR2":
            constraint_category_found = True
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            mode1: str = constraints[key]["mode1"]
            mode2: str = constraints[key]["mode2"]
            k: int = constraints[key]["k"]
            if mode1 == "HA":
                deviation = model.NewIntVar(0, 1000, "deviation")
                total_breaks: int = sum([evaluate_breaks(model, g, t1, slots, parameters) for t1 in teams1])
                if mode2 == "LEQ":
                    model.AddMaxEquality(deviation, [total_breaks - k, 0])
                else:
                    model.AddMaxEquality(deviation, [total_breaks - k, k - total_breaks, 0])
                penalties.append(deviation * penalty_weight)

        # FA2 constraint
        if constraints[key]["category"] == "FA2":
            constraint_category_found = True
            slots: list = constraints[key]["timeslots"]
            teams1: list = constraints[key]["teams1"]
            mode2: str = constraints[key]["mode2"]
            k: int = constraints[key]["k"]
            deviation = model.NewIntVar(0, 5000, "deviation")
            if mode2 == "H":
                for i in teams1:
                    for j in teams1:
                        if i < j:
                            a: list = [0]
                            for s in slots:
                                abs_diff2 = model.NewIntVar(0, 100, 'abs_diff2_{s}')
                                model.AddAbsEquality(abs_diff2, sum(g[(i, t, p)] for t in teams1
                                                for p in range(s + 1) if i != t) -
                                                sum(g[(j, t, p)] for t in teams1 for p in range(s + 1) if j != t))
                                a.append((abs_diff2 - k) * penalty_weight)
                            model.AddMaxEquality(deviation, a)
                            penalties.append(deviation)

        # SE1 constraint
        if constraints[key]["category"] == "SE1":
            constraint_category_found = True
            teams1: list = constraints[key]["teams1"]
            k_min: int = constraints[key]["k_min"]
            for i in teams1:
                for j in teams1:
                    if i < j:
                        deviation = model.NewIntVar(0, 5000, f"deviation_{i}_{j}")
                        abs_d = model.NewIntVar(0, slots_num, f"abs_d_{i}_{j}")
                        model.AddAbsEquality(abs_d, sum((g[(i,j,s)] - g[(j,i,s)]) * s for s in range(slots_num)))
                        model.AddMaxEquality(deviation, [k_min - (abs_d - 1), 0])
                        penalties.append(deviation * penalty_weight)

        if constraint_category_found == False:
            print(f"Error! Soft constraint {constraints[key]['category']} not set up.")

    return penalties
```

Πίνακας 15: CP-SAT: συνάρτηση add_constraints_as_soft

Τέλος, η συνάρτηση *cpsat_task* δέχεται ως όρισμα ένα αρχικό σωματίδιο (κανονικό ή εφικτό), το πλήθος των ομάδων ή των χρονοθυρίδων που θα κρατηθούν ως σταθερές στο μοντέλο, και τον τρόπο (*mode*) της λειτουργίας της.

Η παράμετρος *mode* λαμβάνει 3 τιμές:

- *HARD*: απαιτεί την πλήρη ικανοποίηση των δομικών και ισχυρών περιορισμών.
- *ALL*: απαιτεί την πλήρη ικανοποίηση των δομικών και ισχυρών περιορισμών και
  - την βελτιστοποίηση των ασθενών περιορισμών αν το όρισμα είναι ένα κανονικό μη εφικτό σωματίδιο, ή
  - την αναζήτηση ενός καλύτερου σωματιδίου αν το όρισμα είναι ένα εφικτό σωματίδιο.
- *HARD_AS_SOFT*: απαιτεί την πλήρη ικανοποίηση των δομικών περιορισμών, ενώ διενεργεί βελτιστοποίηση των ισχυρών περιορισμών θεωρώντας τους ως ασθενείς.

Η υλοποίηση της συνάρτησης *cpsat_task* φαίνεται στον ακόλουθο πίνακα:

```
1  def cpsat_task(particle: Particle, teams_fix_num: int, slots_fix_num: int, mode: str, max_runtime = 800, workers =
2  12, show_log = False) -> tuple:
3      """
4      Step procedure for enhancing canonical particle with fix and optimize.
5      Fix some teams or slots of the initial, canonical particle and then,
6      demand satisfaction for structural and hard constraints and
7      minimize the cost of the soft constraints.
8
9      Mode
10     - HARD:         Satisfy only the structural and hard constraints
11     - ALL:          Satisfy all structural and hard constraints and minimize soft constraints.
12                     In the case of a feasible initial particle, do not minimize soft constraints
13                     to save time, and just search for a feasible particle better than the initial.
14     - HARD_AS_SOFT: Satisfy only the structural and minimize hard constraints considered as soft.
15
16     Arguments
17     - particle: The initial particle.
18     - teams_fix_num: The number of teams to be fixed in the initial particle.
19     - slots_fix_num: The number of slots to be fixed in the initial particle.
20     - mode: The type of constraints ("HARD" or "HARD_AS_SOFT" or "ALL").
21     - max_runtime: The maximum runtime after which the search stops.
22     - workers: The maximum number of threads used for searching.
23     - show_log: True to show log; otherwise, False.
24
25     Returns:
26     - status and particle
27     """
28     selected_games: list = []
29
30     # Parameters
31     parameters: dict = particle.parameters
```

```
        isPhased: bool = True if parameters["gamemode"] == "P" else False
        teams_num: int = parameters["teams_num"]
        slots_num: int = parameters["slots_num"]

        # Convert x_is timetable into g timetable
        g_ijs: np.ndarray = convert_xis_to_g(particle.position)

        # Create the model
        model = cp_model.CpModel()

        # Variables
        g = {}
        for i in range(teams_num):
            for j in range(teams_num):
                for s in range(slots_num):
                    if i != j:  # A team cannot play against itself
                        g[(i, j, s)] = model.NewIntVar(0, 1, f'g_{i}_{j}_{s}')
        total_soft_penalty = model.NewIntVar(0, 5000, 'total_soft_penalty')
        total_hard_penalty = model.NewIntVar(0, 5000, 'total_hard_penalty')

        # Fix random number of games
        if teams_fix_num > 0:
            selected_games = fix_entities(g_ijs, teams_fix_num, parameters, 'team')
        elif slots_fix_num > 0:
            selected_games = fix_entities(g_ijs, slots_fix_num, parameters, 'slot')
        for (i, j, s) in selected_games:
            model.Add(g[(i, j, s)] == 1)

        # Add structural constraints
        add_structural_constraints(model, g, teams_num, slots_num, isPhased)

        match mode:
            # Add hard constraints
            case "HARD":
                add_hard_constraints(model, g, parameters)
            # Add hard and soft constraints
            case "ALL":
                add_hard_constraints(model, g, parameters)
                soft_penalties: list = add_constraints_as_soft(model, g, "SOFT", parameters)
                model.Add(total_soft_penalty == sum(soft_penalties))
                # If the initial particle is feasible, demand improvement.
                if particle.is_feasible():
                    model.Add(total_soft_penalty < particle.soft_fitness)
                # Else minimize the soft constraints.
                else:
                    model.Minimize(total_soft_penalty)
            # Add hard constraints as soft
            case "HARD_AS_SOFT":
                hard_penalties: list = add_constraints_as_soft(model, g, "HARD", parameters)
                model.Add(total_hard_penalty == sum(hard_penalties))
                model.Add(total_hard_penalty < particle.hard_fitness)
            case _:
                pass

        # Define the solver
        solver = cp_model.CpSolver()

        # Solver parameters
        if max_runtime > 0:
            solver.parameters.max_time_in_seconds = max_runtime
        solver.parameters.num_workers = workers
        solver.parameters.random_seed = random.randint(10000, 99999)
        solver.parameters.log_search_progress = show_log

        # Solve the model
```

```
97        status = solver.Solve(model)
98
99        # Display the results
100       y_is: np.ndarray = np.full((teams_num, slots_num), fill_value=-1, dtype=int)
101       if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
102           for s in range(slots_num):
103               for i in range(teams_num):
104                   for j in range(teams_num):
105                       if i != j and solver.Value(g[(i, j, s)]):
106                           y_is[i][s] = j
197       return status, Particle(y_is, parameters)
```

Πίνακας 16: CP-SAT: συνάρτηση cpsat_task

7.3 Γεννήτρια παραγωγής σωματιδίων

Για την παραγωγή κανονικών ή εφικτών σωματιδίων που θα αποτελέσουν το αρχικό σμήνος του αλγορίθμου CP-PSO χρησιμοποιούμε την συνάρτηση *cpsat_generator*. Η υλοποίηση αυτής της γεννήτριας με βάση τις παραμέτρους του προβλήματος δίνεται στον Πίνακα 17.

Η συνάρτηση *cpsat_generator* δέχεται όλους τους δομικούς και ισχυρούς περιορισμούς του προβλήματος και αναζητά μία λύση στον χώρο αναζήτησης. Μπορεί να παράξει ένα κανονικό ή ένα εφικτό σωματίδιο. Για τον σκοπό αυτό δέχεται δύο ορίσματα:

- Το όρισμα *only_canonical* που όταν λάβει τιμή True η συνάρτηση παράγει κανονικό σωματίδιο. Στην αντίθετη περίπτωση, η συνάρτηση θα παράξει εφικτό σωματίδιο.
- Τις παραμέτρους του προβλήματος.

Γενικώς, η γεννήτρια παράγει κανονικά σωματίδια σε μικρό χρονικό διάστημα. Για προβλήματα μικρής έκτασης η συνάρτηση έχει επίσης ικανοποιητική απόδοση στην παραγωγή ενός εφικτού σωματιδίου.

```
1   def cpsat_generator(only_canonical: bool, parameters: dict) -> Particle:
2       """
3       Generate a canonical or feasible particle.
4
5       Arguments
6       - only_canonical: True/False for generating a canonical/feasible particle.
7       - parameters: The parameters of the problem.
8
9       Returns:
10      - A particle
11      """
12      # Parameters
13      isPhased: bool = True if parameters["gamemode"] == "P" else False
```

```
    teams_num: int = parameters["teams_num"]
    slots_num: int = parameters["slots_num"]

    # Create the model
    model = cp_model.CpModel()

    # Variables
    # g[i, j, s] == 1 if team i plays a home game against team j at slot s; 0 otherwise.
    g = {}
    for i in range(teams_num):
        for j in range(teams_num):
            for s in range(slots_num):
                if i != j:  # A team cannot play against itself
                    g[(i, j, s)] = model.NewBoolVar(f'g_{i}_{j}_{s}')

    # Add structural constraints
    add_structural_constraints(model, g, teams_num, slots_num, isPhased)

    # Add hard constraints
    if only_canonical == False:
        add_hard_constraints(model, g, parameters)

    # Define the solver
    solver = cp_model.CpSolver()
    solver.parameters.random_seed = random.randint(1, 99999)

    # Solve the model
    status = solver.Solve(model)

    # Display the results
    y_is: np.ndarray = np.full((teams_num, slots_num), fill_value=-1, dtype=int)
    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        for s in range(slots_num):
            for i in range(teams_num):
                for j in range(teams_num):
                    if i != j and solver.Value(g[(i, j, s)]):
                        y_is[i][s] = j
    return Particle(y_is, parameters)
```

Πίνακας 17: Γεννήτρια παραγωγής κανονικών ή εφικτών σωματιδίων

7.4 Αλγόριθμος βελτιστοποίησης CP-SAT Enhancer

Η υλοποίηση του αλγορίθμου βελτιστοποίησης CP-SAT Enhancer που έχουμε περιγράψει στο κεφάλαιο 6.2.2 φαίνεται στον Πίνακα 18. Η συνάρτηση *cpsat_enhancer* λαμβάνει ως όρισμα το σωματίδιο που επιθυμούμε να βελτιωθεί. Το σωματίδιο μπορεί να είναι κανονικό ή εφικτό. Αν πρόκειται για κανονικό σωματίδιο, τότε η βελτιστοποίηση μπορεί να γίνει είτε λαμβάνοντας υπ' όψιν τους δομικούς και τους ισχυρούς περιορισμούς (αντιμετωπίζοντάς τους ως ασθενείς), είτε όλους τους περιορισμούς. Αν πρόκειται για εφικτό σωματίδιο, τότε η μόνη επιλογή είναι να λάβουμε υπ' όψιν όλους τους περιορισμούς.

```python
def cpsat_enhancer(initial_particle: Particle) -> Particle:
    """
    Enhance a canonical or feasible particle with fix and optimize heuristic.

    Teams/slots fixed initial values are the highest possible.
    - If the status is infeasible, then decrease teams/slots fix (after a number
      of static iterations).
    - If the status is unknown, then increase runtime and decrease teams/slots fix
      (after a number of static iterations).
    - If status is feasible, then return the derived particle.
    Arguments:
    - particle: the initial, canonical or feasible particle to be enhanced
    Returns:
    - particle
    """
    q: Particle
    # Set the number of teams/slots to be fixed to their maximum value.
    teams_fix_num: int = initial_particle.parameters["teams_num"] - 1
    slots_fix_num: int = initial_particle.parameters["slots_num"] - 1

    # Max runtime
    max_runtime: int = initial_particle.parameters["cpsat"]["max_runtime"]
    runtime_step: int = initial_particle.parameters["cpsat"]["runtime_step"]

    # Infeasibility count
    count: int  = 0
    while True:
        start_time: float = time.time()
        if initial_particle.is_feasible():
            status, q = cpsat_task(initial_particle, teams_fix_num, slots_fix_num, "ALL", max_runtime)
        else:
            status, q = cpsat_task(initial_particle, teams_fix_num, slots_fix_num,
                                   initial_particle.parameters["cpsat"]["enhancing_mode"], max_runtime)
        end_time: float = time.time()

        # If status is OPTIMAL|FEASIBLE, then return the solution found.
        if status == cp_model.FEASIBLE or status == cp_model.OPTIMAL:
            if teams_fix_num > 0:
                print(f"{'OPTIMAL|FEASIBLE':<20}DT={end_time - start_time:<10.2f}
                                                                teams_fix_num={teams_fix_num:<10}")
            elif slots_fix_num > 0:
                print(f"{'OPTIMAL|FEASIBLE':<20}DT={end_time - start_time:<10.2f}
                                                                slots_fix_num={slots_fix_num:<10}")
            else:
                print(f"{'OPTIMAL|FEASIBLE':<20}DT={end_time - start_time:<10.2f}No fix and optimize.")
            q.print()
            return q

        # If status is unknown, then increase runtime and decrease teams/slots
        if status == cp_model.UNKNOWN:
            if teams_fix_num > 0:
                print(f"{'UNKNOWN':<20}DT={end_time - start_time:<10.2f}teams_fix_num={teams_fix_num:<10}")
            elif slots_fix_num > 0:
                print(f"{'UNKNOWN':<20}DT={end_time - start_time:<10.2f}slots_fix_num={slots_fix_num:<10}")
            else:
                print(f"{'UNKNOWN':<20}DT={end_time - start_time:<10.2f}No fix and optimize.")
            max_runtime += runtime_step

        # If status is infeasible, then decrease teams/slots
        else:
            if teams_fix_num > 0:
                print(f"{'INFEASIBLE':<20}DT={end_time - start_time:<10.2f}teams_fix_num={teams_fix_num:<10}")
            elif slots_fix_num > 0:
                print(f"{'INFEASIBLE':<20}DT={end_time - start_time:<10.2f}slots_fix_num={slots_fix_num:<10}")
            else:
```

```
66                  print(f"{'INFEASIBLE':<20}DT={end_time - start_time:<10.2f}No fix and optimize.")
67
68          count += 1
69          # Explore for a fixed number of times before decreasing fixes
70          if count % initial_particle.parameters["cpsat"]["static_period"] == 0:
71              count = 0
72              if teams_fix_num > 0:
73                  teams_fix_num -= 1
74              elif slots_fix_num > 0:
75                  slots_fix_num -= 1
76              else:
77                  break
78      return initial_particle
```

Πίνακας 18: Αλγόριθμος βελτιστοποίησης CP-SAT Enhancer

7.5 Κυρίως πρόγραμμα

Το κυρίως πρόγραμμα που υλοποιεί τον αλγόριθμο CP-PSO (Σχήμα 21) φαίνεται στον Πίνακα 19. Το πρόγραμμα ξεκινάει διαβάζοντας τις παραμέτρους του προβλήματος (γραμμή 7). Ακολουθεί η φάση ικανοποιησιμότητας, στην οποία γίνεται η παραγωγή των κανονικών ή εφικτών σωματιδίων του σμήνους και αρχικοποιείται το σμήνος (γραμμές 62 – 93). Στην συνέχεια, υπολογίζεται η μέση απόσταση των σωματιδίων και ενημερώνεται το σωματίδιο *gBest* (γραμμές 96 – 98).

Έπειτα ξεκινά η φάση βελτιστοποίησης, δηλαδή μια επαναληπτική διαδικασία αναζήτησης λύσης στον εφικτό και μη εφικτό χώρο αναζήτησης. Στην φάση αυτή εκτελούνται οι εξής διεργασίες:

- Βελτιστοποίηση με Advanced PSO (γραμμές 104 – 110).
- Βελτιστοποίηση με
  - Hill Climbing με CP-SAT (γραμμές 113 – 118).
  - Hill Climbing με πλήρεις τελεστές (γραμμές 121 – 124).

Μετά τον τερματισμό της επαναληπτικής διαδικασίας, το σωματίδιο *gBest* που έχει προκύψει αποθηκεύεται σε τοπικό αρχείο και εκτυπώνονται οι βασικές του πληροφορίες στην οθόνη (γραμμές 134 – 135).

Η υλοποίηση του αλγορίθμου CP-PSO σε γλώσσα Python περιλαμβάνει επίσης την ανάπτυξη κάποιων επιπλέον βοηθητικών συναρτήσεων που παρατίθενται στο Παράρτημα Α.

```python
import numpy as np
import os
from datetime import datetime
from libcppso import Particle,Swarm, cpsat_generator, cpsat_generator, hash_timetable,get_instance_and_parameters

if __name__ == '__main__':
    parameters: dict = get_instance_and_parameters("instance.xml")

    print(f"CP-PSO v1.0")
    print(f"© 2024 Nassos Kranidiotis\n")
    print(f"Execution started at {datetime.now():%d/%m/%Y %H.%M.%S}")
    print(f"------------------------------------------------------------")
    print(f"CP-PSO Parameters:")
    print(f"   Instance name                          : {parameters['instanceName']}")
    print(f"       > number of teams                  : {parameters['teams_num']}")
    print(f"       > number of slots                  : {parameters['slots_num']}")
    print(f"   CP-PSO parameters")
    print(f"       > operation mode                   : {parameters['mode']}")
    print(f"       > particles                        : {parameters['particles_num']}")
    print(f"       > max age                          : {parameters['max_age']}")
    print(f"       > SA exponential coefficient k     : {parameters['k']}")
    print(f"       > inertial probability             : {parameters['inertia_probability']}")
    print(f"       > cognitive probability            : {parameters['cognitive_probability']}")
    print(f"       > social probability               : {parameters['social_probability']}")
    print(f"       > sub-particle fraction            : {parameters['subparticle_fraction']}")
    print(f"       > hard constraint weight           : {parameters['weights']['hard']}")
    print(f"       > soft constraint weights")
    print(f"             - CA1                        : {parameters['weights']['soft']['CA1']}")
    print(f"             - CA2                        : {parameters['weights']['soft']['CA2']}")
    print(f"             - CA3                        : {parameters['weights']['soft']['CA3']}")
    print(f"             - CA4                        : {parameters['weights']['soft']['CA4']}")
    print(f"             - GA1                        : {parameters['weights']['soft']['GA1']}")
    print(f"             - BR1                        : {parameters['weights']['soft']['BR1']}")
    print(f"             - BR2                        : {parameters['weights']['soft']['BR2']}")
    print(f"             - FA2                        : {parameters['weights']['soft']['FA2']}")
    print(f"             - SE1                        : {parameters['weights']['soft']['SE1']}")
    print(f"       > heuristic operators probability")
    print(f"             - swapHomes                  : {parameters['operators']['s_h_weight']}")
    print(f"             - swapOpponents              : {parameters['operators']['s_p_weight']}")
    print(f"             - swapTeams                  : {parameters['operators']['s_t_weight']}")
    print(f"             - swapRounds                 : {parameters['operators']['s_r_weight']}")
    print(f"             - swapMatches                : {parameters['operators']['s_m_weight']}")
    print(f"             - rotateTeams                : {parameters['operators']['r_t_weight']}")
    print(f"   CP-SAT parameters")
    print(f"       > maximum runtime                  : {parameters['cpsat']['max_runtime']}")
    print(f"       > runtime step                     : {parameters['cpsat']['runtime_step']}")
    print(f"       > iterations at fixed games mode   : {parameters['cpsat']['static_period']}")
    print(f"       > enhancing mode (canonical particle): {parameters['cpsat']['enhancing_mode']}")
    print(f"------------------------------------------------------------\n")

    print("STAGE 1 - SATISFIABILITY\n")

    # Declarations
    hash_list: list[str] = []
    hash: str
    timetables_list: list = []
    count: int = 0
    p: Particle
    myswarm: Swarm

    # Generate canonical particles with CP-SAT.
    if parameters["mode"] == "canonical":
        if not os.path.exists("canonical_timetables.npy"):
            while count < parameters['particles_num']:
                p = cpsat_generator(only_canonical = True, parameters = parameters)
```

```
                hash = hash_timetable(p.position)
                if hash not in hash_list:
                    timetables_list.append(p.position)
                    p.save()
                    hash_list.append(hash)
                    count += 1
            np.save("canonical_timetables", timetables_list)
        # Initialize swarm
        myswarm = Swarm(parameters, "canonical_timetables.npy")

    # Generate feasible particles with CP-SAT and/or sub-particles
    if parameters["mode"] == "feasible":
        if not os.path.exists("feasible_timetables.npy"):
            while count < parameters['particles_num']:
                p = cpsat_generator(only_canonical = True, parameters = parameters)
                p.enhance("CP-SAT")
                p.generate_feasible_particle(True)
                if not p.is_feasible():
                    continue
                hash = hash_timetable(p.position)
                if hash not in hash_list:
                    timetables_list.append(p.position)
                    p.save()
                    hash_list.append(hash)
                    count += 1
            np.save("feasible_timetables", timetables_list)
        # Initialize swarm
        myswarm = Swarm(parameters, "feasible_timetables.npy")

    # Estimate average distance and update gBest
    myswarm.average_distance()
    myswarm.update_gbest()
    myswarm.print()

    print("\nSTAGE 2 - OPTIMIZATION\n")
    for cycle in range(20):
        print(f"CYCLE {cycle}\n")

        # Start generations until aging
        print(f"Advanced PSO [{datetime.now():%d/%m/%Y %H.%M.%S}]")
        myswarm.update_swarm()
        myswarm.print()
        myswarm.update_gbest()
        myswarm.gBest.save()
        myswarm.save()

        # Enhance particles with CP-SAT
        print(f"Hill Climbing with CP-SAT [{datetime.now():%d/%m/%Y %H.%M.%S}]")
        myswarm.enhance("CP-SAT")
        myswarm.print()
        myswarm.update_gbest()
        myswarm.gBest.save()
        myswarm.save()

        # Enhance particles with complete operators
        print(f"Hill Climbing with complete operators [{datetime.now():%d/%m/%Y %H.%M.%S}]")
        myswarm.enhance("COMPLETE-OPERATORS")
        myswarm.print()
        myswarm.save()

        # Update gBest
        myswarm.update_gbest()
        myswarm.gBest.save()
```

```
131             # Next cycle
132             cycle += 1
133
134         myswarm.gBest.save()
135         myswarm.gBest.print()
```

Πίνακας 19: Κυρίως πρόγραμμα CP-PSO

□

8 Πειραματικά αποτελέσματα

Η εκτέλεση του αλγορίθμου CP-PSO για την επίλυση των προβλημάτων του διαγωνισμού ITC2021 έγινε σε οικιακούς υπολογιστές με υπολογιστική ισχύ έως 8 vCPUs, μνήμη έως 16 GB RAM και λειτουργικό σύστημα Ubuntu 24.04 LTS. Παράλληλα χρησιμοποιήθηκε ένας μικρός αριθμός από δωρεάν εικονικές μηχανές (*Virtual Machines)* με υπολογιστική ισχύ έως 8 vCPUs στο Google Cloud.

8.1 Σύνολα δεδομένων

Τα προβλήματα του διαγωνισμού ITC2021[3] χωρίζονται σε 3 κατηγορίες: τα αρχικά (*Early*), τα ενδιάμεσα (*Middle*) και τα κατοπινά (*Late*). Κάθε κατηγορία αποτελείται από 15 προβλήματα. Κάθε πρόβλημα έχει μορφοποιηθεί με την βοήθεια της σημειογραφίας RobinX (Van Bulck κ.ά., 2019). Όλα τα προβλήματα αφορούν σε τουρνουά τύπου 2-Round-Robin (2RR) με 6, 16, 18 ή 20 συμμετέχουσες ομάδες, και αγώνες που παίζονται σε 10, 30, 34, 38 χρονοθυρίδες, ενώ σε μερικά από αυτά υπάρχει η απαίτηση το ωρολόγιο πρόγραμμα που θα παραχθεί να είναι σε φάσεις (*phased*). Γενικώς, δεν συμμετέχουν όλοι οι τύποι περιορισμών σε όλα τα προβλήματα του διαγωνισμού.

Επίσης, υπάρχει και ένα πλήθος από 8 δοκιμαστικά προβλήματα που παίζουν τον ρόλο των benchmarks για έναν υποψήφιο αλγόριθμο επίλυσης. Τα πρώτα 5 σημειώνονται με πράσινο χρώμα στην επίσημη ιστοσελίδα του διαγωνισμού ITC2021, γεγονός που δηλώνει την χαμηλή υπολογιστική ισχύ που απαιτείται για την επίτευξη της βέλτιστης λύσης.

Κάθε πρόβλημα χαρακτηρίζεται από δύο τιμές:

- **Το καλύτερο κάτω όριο** (*Best LB*): πρόκειται για ένα θεωρητικό ελάχιστο που μπορεί να επιτύχει η αντικειμενική συνάρτηση υπό την προϋπόθεση ότι υπάρχει μία τέλεια, βέλτιστη λύση. Στα προβλήματα που εξετάζουμε, αντιπροσωπεύει την τιμή της ανεφικτότητας (*infeasibility*) και την ελάχιστη τιμή κόστους ενός ωρολογίου προγράμματος που οφείλεται στους ασθενείς περιορισμούς των οποίων η παραβίαση οδηγεί σε ποινή. Το καλύτερο κάτω όριο είναι συνήθως πολύ αισιόδοξο και είναι δύσκολο να επιτευχθεί.

[3] https://robinxval.ugent.be/ITC2021/instances.php

- **Το καλύτερο άνω όριο** (*Best UB*): πρόκειται για γνωστή τιμή της αντικειμενικής συνάρτησης που συχνά έχει επιτευχθεί με την βοήθεια ευρετικών ή μεταευρετικών μεθόδων επίλυσης που μπορούν να λύσουν το πρόβλημα παράγοντας μία λύση που δεν είναι κατ' ανάγκη βέλτιστη. Το καλύτερο άνω όριο παίζει τον ρόλο του σημείου αναφοράς με το οποίο μπορούμε να αξιολογήσουμε νέες λύσεις.

8.2 Τιμές παραμέτρων

Για την επίλυση κάθε προβλήματος του διαγωνισμού ITC2021 απαιτείται ένα διαφορετικό σύνολο τιμών παραμέτρων του μοντέλου. Μια τυπική εικόνα των παραμέτρων αυτών είναι φαίνεται στον Πίνακα 20 και αναλύονται παρακάτω:

- Η παράμετρος *mode* καθορίζει τον τύπο των σωματιδίων που θα αρχικοποιήσουν το σμήνος. Έτσι, αν επιθυμούμε να αρχικοποιήσουμε το σμήνος με κανονικά (όχι εφικτά) σωματίδια, τότε επιλέγουμε τιμή παραμέτρου *canonical*. Αν θέλουμε να αρχικοποιήσουμε το σμήνος με εφικτά σωματίδια, τότε επιλέγουμε τιμή παραμέτρου *feasible*.
- Το πλήθος των σωματιδίων προσπαθούμε να μην είναι μικρότερο του 16 εκτός αν το πρόβλημα είναι υπολογιστικά πολύ χρονοβόρο και απαιτητικό σε υπολογιστικούς πόρους, οπότε ελαττώνουμε το πλήθος των σωματιδίων σε 8.
- Το μέγιστο πλήθος γενεών, όμως παραμένει ίσο με 10000 σε όλες τις περιπτώσεις. Γενικώς, θεωρείται κοινή παραδοχή ότι ο αλγόριθμος PSO δεν μπορεί να εξερευνήσει καλά τον χώρο αναζήτησης σε πλήθος επαναλήψεων μικρότερο των 10000. Αυτό, ωστόσο δεν έχει αποδειχθεί.
- Ο συντελεστής k καθορίζεται πειραματικά με γνώμονα την ικανότητα του αλγορίθμου να εξερευνήσει τον χώρο αναζήτησης. Αν στα πρώτα στάδια της εκτέλεσής του διαπιστώσουμε ότι η αποδοχή «κακών» λύσεων είναι σπάνια, τότε ο συντελεστής k είναι πολύ μεγάλος και πρέπει να μειωθεί.
- Οι παράμετροι *inertia_probability, cognitive_probability* και *social_probability* είναι οι μεταβλητές P_w, P_c, P_s αντίστοιχα (σχέσεις 76, 77). Οι τιμές των παραμέτρων αυτών που φαίνονται στον Πίνακα 20 έχουν παραχθεί για $\varphi = 2.04$ (Πίνακας 4) με

βάση την εισήγηση των Clerc & Kennedy (2002) που αφορά σε σύγκλιση του Canonical PSO.

- Η παράμετρος *subparticle_fraction* δηλώνει πόσοι από τους ισχυρούς περιορισμούς που ένα μη-εφικτό, κανονικό σωματίδιο δεν ικανοποιεί, θα χρησιμοποιηθούν για την παραγωγή υποσωματιδίων.
- Ακολουθεί το ειδικό βάρος παραβίασης των δομικών και ισχυρών περιορισμών.
- Το ειδικό βάρος παραβίασης των ασθενών περιορισμών ενισχύει το κόστος παραβίασής τους. Έτσι, αν επιθυμούμε να περιορίσουμε μεταλλάξεις σωματιδίων που να παραβιάζουν συγκεκριμένους ασθενείς περιορισμούς, έχουμε την δυνατότητα να αυξήσουμε τα ειδικά βάρη τους.
- Η παράμετρος *max_runtime* καθορίζει τον μέγιστο χρόνο εκτέλεσης του CP-SAT. Αν θέσουμε την τιμή της ίση με το 0, τότε δεν ορίζεται μέγιστος χρόνος εκτέλεσης για τον CP-SAT και ο τελευταίος μπορεί να εκτελεστεί για όσο χρόνο απαιτείται.
- Η παράμετρος *runtime_step* χρησιμοποιείται για να αυξήσει βηματικά τον μέγιστο χρόνο εκτέλεση του CP-SAT.
- Η παράμετρος *enhancing_mode* καθορίζει την μέθοδο βελτιστοποίησης ενός κανονικού σωματιδίου. Λαμβάνει μία από τις ακόλουθες τιμές:
  - *HARD* όπου ικανοποιεί μόνον τους δομικούς και ισχυρούς περιορισμούς.
  - *HARD_AS_SOFT* όπου ικανοποιεί τους δομικούς περιορισμούς και διαχειρίζεται τους ισχυρούς περιορισμούς ως ασθενείς.
  - *ALL* όπου ικανοποιεί τους δομικούς και ισχυρούς περιορισμούς και μειώνει το κόστος των ασθενών περιορισμών σε εφικτό σωματίδιο ή το ελαχιστοποιεί σε κανονικό μη εφικτό σωματίδιο.
- Η παράμετρος *static_period* καθορίζει το πλήθος των δοκιμών που γίνονται για το ίδιο σταθερό πλήθος οντοτήτων (ομάδων/χρονοθυρίδων) του μηχανισμού *fix-and-optimize*.
- Τέλος, οι πιθανότητες επιλογής των ευρετικών τελεστών ισούνται με τα αντίστοιχα βάρη τους (Πίνακας 3).

Παράμετρος	Τιμή	Περιγραφή
mode	canonical	Μοντέλο αρχικοποίησης σμήνους (canonical ή feasible)
particles_num	16	Πλήθος σωματιδίων σμήνους
max_age	10000	Μέγιστο πλήθος γενεών
k	6.7	Εκθετικός συντελεστής στην σχέση $P = e^{-kf_w/T}$
inertia_probability	0.1969	Πιθανότητα μετάλλαξης (συντελεστής αδράνειας)
cognitive_probability	0.4015	Πιθανότητα διασταύρωσης με το pBest (νοητικός συντελεστής)
social_probability	0.4015	Πιθανότητα διασταύρωσης με το gBest (κοινωνικός συντελεστής)
subparticle_fraction	3	Πλήθος violated hard constraints στην παραγωγή υποσωματιδίων
Weights:		
structural	100000	Ειδικό βάρος παραβίασης δομικών περιορισμών
hard	15	Ειδικό βάρος παραβίασης ισχυρών περιορισμών
Soft:		
CA1	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού CA1
CA2	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού CA2
CA3	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού CA3
GA1	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού GA1
BR1	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού BR1
BR2	1	Ειδικό βάρος παραβίασης ασθενούς περιορισμού BR2
FA2	5	Ειδικό βάρος παραβίασης ασθενούς περιορισμού FA2
SE1	5	Ειδικό βάρος παραβίασης ασθενούς περιορισμού SE1
CPSAT:		
max_runtime	1600	Μέγιστος χρόνος εκτέλεσης του CP-SAT
runtime_step	1	Βήμα αύξησης χρόνου εκτέλεσης του CP-SAT
enhancing_mode	“HARD_AS_SOFT”	Τρόποι λειτουργίας βελτιστοποίησης κανονικών σωματιδίων
static_period	10	Πλήθος επαναλήψεων με στατικές τιμές ομάδων/χρονοθυρίδων
Operators:		
s_h_weight	0.20	Πιθανότητα επιλογής του τελεστή swap_homes
s_p_weight	0.23	Πιθανότητα επιλογής του τελεστή swap_opponents
s_t_weight	0.15	Πιθανότητα επιλογής του τελεστή swap_teams
s_r_weight	0.14	Πιθανότητα επιλογής του τελεστή swap_rounds
s_m_weight	0.16	Πιθανότητα επιλογής του τελεστή swap_matches
r_t_weight	0.12	Πιθανότητα επιλογής του τελεστή rotate_teams

Πίνακας 20: Ενδεικτικές τιμές παραμέτρων αλγορίθμου CP-PSO.

8.3 Αποτελέσματα

Κάθε αποτέλεσμα (objective) της διαδικασίας επίλυσης ενός προβλήματος χρονοπρογραμματισμού αθλητικών αγώνων του διαγωνισμού ITC2021 δίνεται στην μορφή (α, β) όπου

- α είναι το κόστος παραβίασης των ισχυρών περιορισμών ($\alpha \geq 0$), και
- β είναι το κόστος παραβίασης των ασθενών περιορισμών ($\beta \geq 0$).

Τα αποτελέσματα (objectives) μπορούν να αντιστοιχούν, επομένως είτε σε κανονικά, είτε σε εφικτά σωματίδια. Ένα αποτέλεσμα (objective) είναι λύση ενός προβλήματος του διαγωνισμού ITC2021, αν αντιστοιχεί σε εφικτό σωματίδιο και τότε θα είναι της μορφής $(0, \beta)$.

Στους πίνακες των αποτελεσμάτων που ακολουθούν, με έντονη γραφή σημειώνονται οι λύσεις που έχουν επιτύχει κόστος ίσο με την καλύτερη γνωστή λύση μέχρις στιγμής. Με αστερίσκο σημειώνονται τα προβλήματα που ο αλγόριθμος CP-PSO βρήκε την καλύτερη γνωστή βέλτιστη λύση, αλλά με διαφορετικό ωρολόγιο πρόγραμμα από αυτό που δημοσιεύεται στα αποτελέσματα του διαγωνισμού. Οι καλύτερες γνωστές λύσεις κάθε προβλήματος έχουν εξαχθεί από την επίσημη ιστοσελίδα του διαγωνισμού ITC2021 μέχρι την ημερομηνία υποβολής της παρούσας εργασίας.

8.3.1 Benchmarks

Ο αλγόριθμος CP-PSO εκτελέστηκε αρχικώς στα 5 πρώτα δοκιμαστικά προβλήματα του διαγωνισμού ITC2021, τα οποία και πέρασε με επιτυχία καταφέρνοντας να βρει την βέλτιστη λύση. Τα προβλήματα αυτά σημειώνονται με πράσινο χρώμα στην επίσημη ιστοσελίδα του διαγωνισμού και απαιτούν μικρή υπολογιστική ισχύ (όπως και η δική μας διαθέσιμη) για την επίλυσή τους.

Στον Πίνακα 21 δίνονται τα αποτελέσματα αυτά. Όπου υπάρχει αστερίσκος δίπλα στην ευρεθείσα λύση υποδηλώνει ότι το ωρολόγιο πρόγραμμα που βρέθηκε είναι διαφορετικό από το ωρολόγιο πρόγραμμα που δίνεται στην βάση δεδομένων του διαγωνισμού ITC2021, μολονότι η τιμή της αντικειμενικής συνάρτησης είναι η ίδια.

Instance	Best Lower Bound	Best Upper Bound	ITC2021 best known solution	CP-PSO solution
Test 1	(0, 1066)	(0, 1066)	(0, 1066)	**(0, 1066)**
Test 2	(0, 176)	(0, 176)	(0, 176)	**(0, 176) ***
Test 3	(0, 1253)	(0, 1253)	(0, 1253)	**(0, 1253)**
Test 4	(0, 4535)	(0, 4535)	(0, 4535)	**(0, 4535)**
Test 5	(0, 2)	(0, 2)	(0, 2)	**(0, 2) ***

Πίνακας 21: Αποτελέσματα εκτέλεσης του αλγορίθμου CP-PSO στα 5 πρώτα δοκιμαστικά προβλήματα του διαγωνισμού ITC2021. Με αστερίσκο σημειώνονται τα προβλήματα που ο αλγόριθμος CP-PSO βρήκε την καλύτερη γνωστή βέλτιστη λύση αλλά με διαφορετικό ωρολόγιο πρόγραμμα.

8.3.2 Προβλήματα διαγωνισμού ITC2021

Ο αλγόριθμος CP-PSO δοκιμάστηκε σε όλα τα προβλήματα του διαγωνισμού ITC2021. Τα αποτελέσματα (objectives) από την εκτέλεση του αλγορίθμου CP-PSO φαίνονται στους Πίνακες 22, 23 και 24 για τα προβλήματα Early, Middle και Late αντιστοίχως. Κάθε αποτέλεσμα συγκρίνεται με:

- την καλύτερη λύση που έχει βρεθεί μέχρι στιγμής και είναι δημοσιευμένη στην επίσημη ιστοσελίδα του διαγωνισμού ITC2021.
- το αποτέλεσμα των Dimitsas κ.ά. (2022), οι οποίοι επιχειρούν την λύση των προβλημάτων του διαγωνισμού ITC2021 με έναν υβριδικό αλγόριθμο που βασίζεται στον αλγόριθμο Simulated Annealing και στον CP-SAT.
- το αποτέλεσμα των Rosati κ.ά. (2022), που χρησιμοποιούν έναν αλγόριθμο βασισμένο στον αλγόριθμο Simulated Annealing.

Τα αρχεία των αποτελεσμάτων (λύσεων και μη) σε μορφή XLM βρίσκονται στο αποθετήριό μας στο github[4]. *Η ορθότητα όλων των λύσεων έχει επιβεβαιωθεί χρησιμοποιώντας τον Solution Validator[5] που παρέχεται από τον διαγωνισμό ITC2021.*

Στο Παράρτημα Β δίνονται όλα τα αποτελέσματα αναλυτικά.

[4] https://github.com/ankranidiotis/cppso.git
[5] https://robinxval.ugent.be/ITC2021/validator.php

Instance	**Best Lower Bound**	**Best Upper Bound**	**ITC2021 best known solution**	**Dimitsas et al. (2022) objective** (run on a 64 vCPU machine)	**Rosati et al. (2022) objective** (run on a 64 vCPU machine)	**CP-PSO objective** (run on a 8 vCPU machine)
Early 1	(0, 1)	(0, 362)	(0. 362)	(0, 512)	(0, 423)	(0, 1603)
Early 2	(0, 0)	(0, 144)	(0, 144)	(0, 266)	(0, 318)	(0, 713)
Early 3	(0, 49)	(0, 934)	(0, 934)	(0, 1354)	(0, 1068)	(0, 1419)
Early 4	(0, 0)	(0, 430)	(0, 430)	(6, —)	(0, 556)	(9, —)
Early 5	(0, 270)	(0, 3127)	(0, 3127)	(5, —)	(0, 4117)	(36, —)
Early 6	(0, 607)	(0, 3287)	(0, 3287)	(0, 3957)	(0, 3927)	(0, 5630)
Early 7	(0, 1296)	(0, 4744)	(0, 4744)	(0, 9644)	(0, 5205)	(0, 10818)
Early 8	(0, 213)	(0, 1051)	(0, 1051)	(0, 1614)	(0, 1051)	(0, 1994)
Early 9	(0, 0)	(0, 56)	(0, 56)	(0, 448)	(0, 132)	(0, 1613)
Early 10	(0, 331)	(0, 3400)	(0, 3400)	(32, —)	(0, 4986)	(34, —)
Early 11	(0, 348)	(0, 4381)	(0, 4381)	(0, 8189)	(0, 4526)	(0, 11442)
Early12	(0, 0)	(0, 315)	(0, 315)	(0, 1025)	(0, 1010)	(0, 1445)
Early 13	(0, 2)	(0, 121)	(0, 121)	(0, 380)	(0, 173)	(0, 730)
Early 14	(0, 1)	(0, 4)	(0, 4)	(0, 63)	(0, 63)	(0, 1392)
Early 15	(0, 485)	(0, 2955)	(0, 2955)	(0, 4470)	(0, 3556)	(0, 6681)

Πίνακας 22: Αποτελέσματα εκτέλεσης του αλγορίθμου CP-PSO στα Early προβλήματα του διαγωνισμού ITC2021.

Instance	**Best Lower Bound**	**Best Upper Bound**	**ITC2021 best known solution**	**Dimitsas et al. (2022) objective** (run on a 64 vCPU machine)	**Rosati et al. (2022) objective** (run on a 64 vCPU machine)	**CP-PSO objective** (run on a 8 vCPU machine)
Middle 1	(0, 2955)	(0, 5177)	(0, 5177)	(17, —)	(0, 5657)	(12, —)
Middle 2	(0, 2984)	(0, 7381)	(0, 7381)	(48, —)	(5, —)	(35, —)
Middle 3	(0, 3378)	(0, 9315)	(0, 9315)	(0, 12170)	(0, 9542)	(0, 11809)
Middle 4	(0, 7)	(0, 7)	(0, 7)	(0, 7)	(0, 16)	**(0, 7) ***
Middle 5	(0, 47)	(0, 279)	(0, 279)	(0, 732)	(0, 510)	(0, 1029)
Middle 6	(0, 24)	(0, 1090)	(0, 1090)	(0, 1900)	(0, 1701)	(0, 1920)
Middle 7	(0, 27)	(0, 1780)	(0, 1780)	(0, 2792)	(0, 2203)	(0, 2672)
Middle 8	(0, 2)	(0, 129)	(0, 129)	(0, 301)	(0, 136)	(0, 447)
Middle 9	(0, 0)	(0, 415)	(0, 415)	(0, 1015)	(0, 640)	(0, 2185)
Middle 10	(0, 4)	(0, 1228)	(0, 1228)	(1, —)	(0, 1357)	(0, 2211)
Middle 11	(0, 345)	(0, 2177)	(0, 2177)	(0, 2956)	(0, 2696)	(0, 3638)
Middle 12	(0, 1)	(0, 597)	(0, 597)	(0, 1596)	(0, 950)	(0, 3291)
Middle 13	(0, 0)	(0, 211)	(0, 211)	(0, 780)	(0, 362)	(0, 4715)
Middle 14	(0, 0)	(0, 1140)	(0, 1140)	(0, 1619)	(0, 1172)	(0, 2889)
Middle 15	(0, 1)	(0, 462)	(0, 462)	(0, 1833)	(0, 985)	(0, 6290)

Πίνακας 23: Αποτελέσματα εκτέλεσης του αλγορίθμου CP-PSO στα Middle προβλήματα του διαγωνισμού ITC2021. Με αστερίσκο σημειώνονται τα προβλήματα που ο αλγόριθμος CP-PSO βρήκε την καλύτερη γνωστή βέλτιστη λύση αλλά με διαφορετικό ωρολόγιο πρόγραμμα.

Instance	**Best Lower Bound**	**Best Upper Bound**	**ITC2021 best known solution**	**Dimitsas et al. (2022) objective** (run on a 64 vCPU machine)	**Rosati et al. (2022) objective** (run on a 64 vCPU machine)	**CP-PSO objective** (run on a 8 vCPU machine)
Late 1	(0, 1103)	(0, 1919)	(0, 1919)	(0, 2234)	(0, 2021)	(0, 3115)
Late 2	(0, 2818)	(0, 5379)	(0, 5379)	(0, 5680)	(0, 5715)	(0, 6434)
Late 3	(0, 416)	(0, 2369)	(0, 2369)	(0, 3004)	(0, 2457)	(0, 2833)
Late 4	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	**(0, 0) ***
Late 5	(0, 398)	(0, 1849)	(0, 1849)	(39, —)	(0, 2341)	(29, —)
Late 6	(0, 6)	(0, 872)	(0, 872)	(0, 1440)	(0, 930)	(0, 1648)
Late 7	(0, 5)	(0, 1558)	(0, 1558)	(0, 3009)	(0, 1765)	(0, 7082)
Late 8	(0, 78)	(0, 934)	(0, 934)	(0, 1375)	(0, 997)	(0, 1971)
Late 9	(0, 3)	(0, 498)	(0, 498)	(0, 1108)	(0, 715)	(0, 2426)
Late 10	(0, 1)	(0, 1786)	(0, 1786)	(6, —)	(0, 2571)	(21, —)
Late 11	(0, 0)	(0, 201)	(0, 201)	(0, 511)	(0, 207)	(0, 796)
Late 12	(0, 203)	(0, 3226)	(0, 3226)	(0, 7218)	(0, 3944)	(0, 9566)
Late 13	(0, 7)	(0, 1813)	(0, 1813)	(0, 3576)	(0, 1862)	(0, 7810)
Late 14	(0, 7)	(0, 1140)	(0, 1140)	(0, 1650)	(0, 1202)	(0, 1966)
Late 15	(0, 0)	(0, 0)	(0, 0)	(0, 80)	(0, 60)	(0, 1280)

Πίνακας 24: Αποτελέσματα εκτέλεσης του αλγορίθμου CP-PSO στα Late προβλήματα του διαγωνισμού ITC2021.

8.4 Συμπεράσματα και αξιολόγηση

Ο αλγόριθμος CP-PSO δοκιμάστηκε στο σύνολο των προβλημάτων του διαγωνισμού ITC2021. Τα συγκριτικά αποτελέσματα απόδοσης εύρεσης λύσης στα προβλήματα του διαγωνισμού ITC2021 φαίνονται στον Πίνακα 25.

No.	Αλγόριθμος	Λύσεις που βρέθηκαν από το σύνολο των 45 προβλημάτων του διαγωνισμού ITC2021	Ποσοστό επιτυχίας εύρεσης λύσης
1	Rosati et al. (2022)	44	97.8%
2	**CP-PSO**	**38**	**84.4%**
3	Dimitsas et al. (2022)	37	82.2%

Πίνακας 25: Συγκριτικά αποτελέσματα απόδοσης εύρεσης λύσης.

Τα αποτελέσματα που βρήκαμε από την επίλυση των δημοσιευμένων προβλημάτων του διαγωνισμού ITC2021 δεν αντιπροσωπεύουν την πλήρη δυναμική του αλγορίθμου CP-PSO, διότι η επίλυση έγινε σε περιορισμένα πλαίσια χρόνου και με πολύ μικρή υπολογιστική ισχύ (έως 8 vCPU) σε σχέση με την υπολογιστική ισχύ που είχαν στην διάθεσή τους οι άλλοι ερευνητές (64 vCPU). Η πολύ μικρή υπολογιστική ισχύ των συστημάτων που χρησιμοποιήθηκαν κατά την επίλυση των προβλημάτων μας ανάγκασε να διακόπτουμε την διαδικασία επίλυσης, προκειμένου να διαθέσουμε τους υπολογιστικούς πόρους σε κάποιο άλλο πρόβλημα. Έτσι, για κάθε πρόβλημα του διαγωνισμού ITC2021 ο αλγόριθμος CP-PSO εκτέλεσε το πολύ έναν κύκλο, ενώ το επιθυμητό θα ήταν τουλάχιστον τρεις. Αυτό είχε ως αποτέλεσμα, οι λύσεις που δημοσιεύουμε να μην είναι οι καλύτερες δυνατές και η σύγκριση με τα αποτελέσματα των άλλων ερευνητών να μην γίνεται σε αντικειμενική βάση.

Παρ' όλα αυτά, ο αλγόριθμος CP-PSO στα προβλήματα Test 2, Test 5, Middle 4 και Late 4 του διαγωνισμού ITC2021 βρήκε την δημοσιευμένη από τον διαγωνισμό βέλτιστη τιμή της αντικειμενικής συνάρτησης του προβλήματος αλλά με διαφορετικό ωρολόγιο πρόγραμμα από αυτό του διαγωνισμού. Αυτό το γεγονός επιβεβαιώνει την δυσκολία των προβλημάτων αυτών που δικαίως κατατάσσονται στην οικογένεια των NP-hard προβλημάτων, διότι σημαίνει πως:

- είτε η λύση που βρήκαμε είναι ένα από πολλά ολικά βέλτιστα του προβλήματος,
- είτε η λύση που βρήκαμε είναι ένα τοπικό βέλτιστο, όπως και η αντίστοιχη του διαγωνισμού, από πολλά τοπικά βέλτιστα του προβλήματος και το ολικό βέλτιστο δεν έχει ανακαλυφθεί ακόμα.

Γενικώς, όταν ο χώρος των λύσεων έχει πολλά ολικά ή τοπικά βέλτιστα, το πρόβλημα είναι ιδιαίτερα δύσκολο και η εύρεση μίας καλής λύσης πολύ απαιτητική.

Λαμβάνοντας υπ' όψιν τα παραπάνω, ο αλγόριθμος που προτείνουμε είναι πολύ υποσχόμενος. Συνδυάζοντας την αποδεδειγμένη δυναμική του αλγορίθμου υπολογιστικής νοημοσύνης PSO με την ευρετική δύναμη του Constraint Programming, επιλύει το πρόβλημα σε δύο φάσεις: την φάση ικανοποιησιμότητας κατά την οποία παράγει αρχικά (κανονικά ή εφικτά) σωματίδια, και την φάση της βελτιστοποίησης στην οποία επιχειρεί να βελτιστοποιήσει τα σωματίδια αυτά εξερευνώντας τον χώρο αναζήτησης. Τα πειραματικά αποτελέσματα από την επίλυση των δοκιμαστικών προβλημάτων του ITC2021 δείχνουν ότι ο αλγόριθμος λειτουργεί άρτια και είναι σε θέση να επιτύχει πολύ καλύτερα αποτελέσματα στα προβλήματα του διαγωνισμού, αν διαθέτει περισσότερη υπολογιστική ισχύ.

8.4.1 Περιορισμοί της τρέχουσας έρευνας

Η τρέχουσα έρευνα περιορίστηκε από την πολύ μικρή υπολογιστική ισχύ (έως 8 vCPU) που είχαμε στην διάθεσή μας, η οποία σε συνδυασμό με τον περιορισμένο χρόνο μέχρι την υποβολή της παρούσας εργασίας, δεν μας επέτρεψε να εμβαθύνουμε στην επίλυση των προβλημάτων του διαγωνισμού ITC2021 επιτυγχάνοντας καλύτερες λύσεις, μολονότι χρησιμοποιούσαμε παραλληλισμό και αξιοποιούσαμε στο έπακρο την υπολογιστική ισχύ του συστήματός μας.

Ένας άλλος περιορισμός στην τρέχουσα έρευνα ήταν η πρόσβαση σε διαθέσιμους CP solvers. Στην παρούσα εργασία επιλέξαμε τον αλγόριθμο CP-SAT, ο οποίος ανήκει στην οικογένεια εργαλείων OR-Tools της Google και είναι ανοικτού κώδικα (*open source*). Η Google προσφέρει δωρεάν και χωρίς κανένα περιορισμό χρήσης τον συγκεκριμένο αλγόριθμο. Ωστόσο, υπάρχουν και άλλοι solvers, όπως ο IBM CPPLEX και ο Gurobi που δεν είναι ανοικτού κώδικα και δεν είχαμε την δυνατότητα να δοκιμάσουμε την απόδοσή τους.

8.4.2 Προτάσεις για μελλοντική έρευνα

Προκειμένου να αντιμετωπιστεί ο υψηλός υπολογιστικός χρόνος που απαιτείται για την επίλυση ενός προβλήματος χρονοπρογραμματισμού του διαγωνισμού ITC2021 από τον αλγόριθμο CP-PSO, μία μελλοντική έρευνα θα μπορούσε να αντικαταστήσει την χρήση του CP-SAT ως αλγορίθμου βελτιστοποίησης των σωματιδίων του σμήνους με άλλους CP solvers. Τέτοιες εναλλακτικές επιλογές θα μπορούσαν να είναι ο IBM CPPLEX ή ο Gurobi οι οποίοι δεν είναι ανοικτού κώδικα, αλλά παρέχονται για ακαδημαϊκή χρήση και ως εμπορικά προγράμματα είναι πιθανόν να αποδίδουν καλύτερα.

Μια άλλη πρόταση για μελλοντική έρευνα θα μπορούσε να αφορά στον τρόπο με τον οποίον ενημερώνεται κάθε σωματίδιο του σμήνους από το σωματίδιο gBest. Όπως είδαμε στο κεφάλαιο 2.3.2 υπάρχουν δύο τρόποι για να γίνει αυτή η ενημέρωση: ο Global PSO και ο Local PSO. Στην παρούσα μελέτη υιοθετήθηκε ο πρώτος τρόπος. Όμως, με την χρήση της μετρικής που ορίσαμε στο κεφάλαιο 5.5 και τον καθορισμό ενός πάνω ορίου απόστασης για την οριοθέτηση της γειτονιάς κάθε σωματιδίου, μπορεί να εξεταστεί η απόδοση του δεύτερου τρόπου ενημέρωσης (Local PSO).

Τέλος, θα ήταν σκόπιμο να αξιολογηθεί σε αντικειμενική βάση ο αλγόριθμος CP-PSO επιλύοντας τα προβλήματα του διαγωνισμού ITC2021 χρησιμοποιώντας υπολογιστική ισχύ (64 vCPU) ανάλογη με εκείνη των άλλων ερευνητών.

□

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

Anagnostopoulos, A., Michel, L., Van Hentenryck, P., & YA, V. (2006). A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, *9*, 177–193. https://doi.org/10.1007/s10951-006-7187-8

Ball, B. C., & Webster, D. B. (1977). Optimal Scheduling for Even-Numbered Team Athletic Conferences. *A I I E Transactions*, *9*(2), 161–169. https://doi.org/10.1080/05695557708975138

Chu, S.-C., Chen, Y.-T., & Ho, J.-H. (2006). Timetable Scheduling Using Particle Swarm Optimization. *First International Conference on Innovative Computing, Information and Control - Volume I (ICICIC'06)*, *3*, 324–327. https://doi.org/10.1109/ICICIC.2006.541

Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE*, *Trans Evol Comput 6(1)*, 58–73.

Costa, F., Urrutia, S., & Ribeiro, C. (2012). An ILS heuristic for the traveling tournament problem with predefined venues. *Annals OR*, *194*, 137–150. https://doi.org/10.1007/s10479-010-0719-9

Dimitsas, A., Gogos, C., Valouxis, C., Tzallas, A., & Alefragis, P. (2022). A pragmatic approach for solving the sports scheduling problem. *Proc. 13th Int. Conf. Pract. Theory Autom. Timatabling, PATAM*, *3*, 195–207.

Easton, K., Nemhauser, G., & Trick, M. (2001). The Traveling Tournament Problem Description and Benchmarks. Στο *Lecture Notes in Computer Science* (τ. 2239, σ. 584). https://doi.org/10.1007/3-540-45578-7_43

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43. https://doi.org/10.1109/MHS.1995.494215

Fonseca, G. H. G., & Toffolo, T. A. M. (2022). A fix-and-optimize heuristic for the ITC2021 sports timetabling problem. *Journal of Scheduling*, *25*(3), 273–286. https://doi.org/10.1007/s10951-022-00738-6

Guo, S., Wang, J., & Guo, M. (2020). Z-Shaped Transfer Functions for Binary Particle Swarm Optimization Algorithm. *Computational Intelligence and Neuroscience*, *2020*, 1–21. https://doi.org/10.1155/2020/6502807

Johnson, D., Aragon, C., McGeoch, L., & Schevon, C. (1989). Optimization by Simulated Annealing: An Experimental Evaluation. Part I, Graph Partitioning. *Operations Research*, *37*, 865–892. https://doi.org/10.1287/opre.37.6.865

Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, *5*, 4104–4108. 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. https://doi.org/10.1109/ICSMC.1997.637339

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science (New York, N.Y.)*, *220*, 671–680. https://doi.org/10.1126/science.220.4598.671

Luke, S. (2013). *Essentials of Metaheuristics* (2nd έκδ.). Lulu. http://cs.gmu.edu/~sean/book/metaheuristics/

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114

Parsopoulos, K. E. (2018). Particle Swarm Methods. Στο R. Martí, P. M. Pardalos, & M. G. C. Resende (Επιμ.), *Handbook of Heuristics* (σσ. 639–685). Springer International Publishing. https://doi.org/10.1007/978-3-319-07124-4_22

Polya, G. (1945). *How to Solve It* (2nd έκδ.). Princeton University Press.

Ribeiro, C. C., & Urrutia, S. (2004). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, *179*(3), 775–787. https://doi.org/10.1016/j.ejor.2005.03.061

Rosati, R. M., Petris, M., Di Gaspero, L., & Schaerf, A. (2022). Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021. *Journal of Scheduling*, *25*(3), 301–319. https://doi.org/10.1007/s10951-022-00740-y

Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, *3*(3), 175–184. https://doi.org/10.1093/comjnl/3.3.175

Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 69–73. https://doi.org/10.1109/ICEC.1998.699146

Van Bulck, D., Goossens, D., Schönberger, J., & Guajardo, M. (2019). RobinX: A three-field classification and unified data format for round-robin sports timetabling. *European Journal of Operational Research*, *280*. https://doi.org/10.1016/j.ejor.2019.07.023

Παράρτημα Α

Στο παράρτημα αυτό παρατίθενται κάποιες επιπλέον βοηθητικές συναρτήσεις που έχουν χρησιμοποιηθεί στην υλοποίηση του αλγορίθμου CP-PSO σε γλώσσα Python.

```python
# region CONVERTERS

def convert_xis_to_g(x_is: np.ndarray) -> np.ndarray:
    """
    Convert x[i, s] to g[i,j,s]

    Arguments:
    - timetable in x_is encoding

    Returns:
    - timetable in g encoding
    """
    teams_num: int = len(x_is)
    slots_num: int = len(x_is[0])
    g_ijs: np.ndarray = np.zeros((teams_num, teams_num, slots_num), dtype=int)
    for team in range(teams_num):
        for slot in range(slots_num):
            opponent = x_is[team, slot]
            if opponent != -1:
                g_ijs[team, opponent, slot] = 1
    return g_ijs


def convert_xij_to_g(x_ij: np.ndarray, slots_num: int) -> np.ndarray:
    """
    Convert x[i,j] to g[i,j,s]

    Arguments:
    - timetable in x_ij encoding

    Returns:
    - timetable in g encoding
    """
    teams_num: int  = len(x_ij)
    g_ijs: np.ndarray = np.zeros((teams_num, teams_num, slots_num), dtype=int)
    for i in range(teams_num):
        for j in range(teams_num):
            s = x_ij[i, j]
            if s != -1:
                g_ijs[i, j, s] = 1
    return g_ijs


def convert_xij_to_xis(x_ij: np.ndarray, slots_num: int) -> np.ndarray:
    """
    Convert x[i,j] to x[i,s]

    Arguments:
    - timetable in x_ij encoding

    Returns:
    - timetable in x_is encoding
    """
```

```
54          teams_num: int = len(x_ij)
55          x_is: np.ndarray = np.full((teams_num, slots_num), fill_value = -1, dtype=int)
56          for i in range(teams_num):
57              for j in range(teams_num):
58                  if i != j:
59                      s = x_ij[i, j]
60                      x_is[i, s] = j
61          return x_is
62
63
64      def convert_xis_to_xij(x_is: np.ndarray) -> np.ndarray:
65          """
66          Convert x[i,s] to x[i,j]
67
68          Arguments:
69          - timetable in x_is encoding
70
71          Returns:
72          - timetable in x_ij encoding
73          """
74          teams_num: int  = len(x_is)
75          slots_num: int  = len(x_is[0])
76          x_ij: np.ndarray = np.full((teams_num, teams_num), fill_value = -1,  dtype=int)
77          for i in range(teams_num):
78              for s in range(slots_num):
79                  if x_is[i][s] != -1:
80                      j = x_is[i][s]
81                      x_ij[i, j] = s
82          return x_ij
83
84
85      def convert_xis_to_xs(x_is: np.ndarray) -> list:
86          """
87          Convert x[i,s] to x[s]
88
89          Arguments:
90          - timetable in x_is encoding
91
92          Returns:
93          - timetable in x_s encoding
94          """
95          teams_num: int = len(x_is)
96          slots_num: int = len(x_is[0])
97          x_s: list = []
98          for s in range(slots_num):
99              matches: list = []
100             for i in range(teams_num):
101                 if x_is[i, s] != -1:
102                     matches.append([i, x_is[i, s]])
103             x_s.append(matches)
104         return x_s
105
106
107     def convert_xs_to_xis(x_s: list, teams_num: int) -> np.ndarray:
108         """
109         Convert x[s] to x[i, s]
110
111         Arguments:
112         - timetable in g encoding
113
114         Returns:
115         - timetable in x_is encoding
116         """
116         slots_num: int  = len(x_s)
117         x_is: np.ndarray = np.full([teams_num, slots_num], fill_value = -1, dtype = int)
```

```
118         for s, group in enumerate(x_s):
119             for match in group:
120                 i = match[0]
121                 j = match[1]
122                 x_is[i, s] = j
123         return x_is
124 
125 
126     def convert_str_to_meetings(inputString: str) -> list:
127         """
128         Converts a string of the form "a,b;c,d" into a list [(a,b), (c,d)]
129 
130         Arguments:
131         - string
132 
133         Returns:
134         - list
135         """
136         elementList: list = []
137         # Split the string based on the semicolon
138         semicolonSplit = inputString.split(';')
139         for part in semicolonSplit:
140             # Split each semicolon-part based on the comma and convert the parts to integers
141             # if the part length is not zero
142             if len(part) > 0:
143                 commaSplit = part.split(',')
144                 elementList.append((int(commaSplit[0]), int(commaSplit[1])))
145         return elementList
146 
147 
148     def convert_g_to_xis(g: np.ndarray) -> np.ndarray:
149         """
150         Convert g[i, j, s] into x[i, s]
151 
152         Arguments:
153         - timetable in g encoding
154 
155         Returns:
156         - timetable in x_is encoding
157         """
158         x_is: np.ndarray = np.full((len(g), len(g[0][0])), fill_value=-1, dtype=int)
159         for i in range(len(g)):
160             for j in range(len(g)):
161                 for s in range(len(g[0][0])):
162                     if g[i, j, s] == 1:
163                         x_is[i, s] = j
164         return x_is
165 
166 
167     # endregion
168 
169     # region AUXILIARY FUNCTIONS
170 
171     def hash_timetable(x_is: np.ndarray) -> str:
172       """
173       Return a hash of a timetable x[i,s].
174 
175       Arguments:
176       - x_is: timetable
177 
178       Returns:
179       - hash: str
180       """
181       hash_object = hashlib.sha256()
182       for i in range(len(x_is)):
```

```
183         for s in range(len(x_is[0])):
184             hash_object.update(str(x_is[i][s]).encode('utf-8'))
185       return hash_object.hexdigest()
186
187
188   def swap_columns(x_is: np.ndarray, s1: int, s2: int) -> np.ndarray:
189       """
190       Swap the s1 column with the s2 column in a 2D timetable.
191
192       Arguments:
193       - x_is: timetable
194       - s1: slot1
195       - s2: slot2
196
197       Returns:
198       - timetable: np.ndarray
199       """
200       y_is: np.ndarray = copy.deepcopy(x_is)
201       for i in range(len(y_is)):
202           j1: int = y_is[i][s1]
203           j2: int = y_is[i][s2]
204           if s1 != s2:
205               y_is[i][s1] = j2
206               y_is[i][s2] = j1
207       return y_is
208
209
210   def get_random_sample(samples_pool)-> tuple:
211       """
212       Draw two random samples from a sample pool.
213
214       Arguments:
215       - sample_pool: list of samples
216
217       Returns:
218       - tuple
219       """
220       # Draw random selections t1, t2 different from each other
221       t1: int = random.randint(0, samples_pool - 1)
222       t2: int = random.randint(0, samples_pool - 1)
223       while t1 == t2:
224           t2 = random.randint(0, samples_pool - 1)
225       return (t1, t2)
226
227
228   def y_function(i: int, j: int, s1: int, s2: int, g_ijs: np.ndarray) -> int:
229       """
230       If teams (i,j) meet at slot s1 and then at slot s2 and not in between,
231       then return 1; else return 0
232       """
233       # Check if i,j do not play each other in between s1 and s2
234       gamesPlayedInBetween: int = 0
235       for s in range(s1 + 1, s2):
236           gamesPlayedInBetween += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
237       # Check if i,j play each other at s1 and s2
238       if (int(g_ijs[i][j][s1]) == 1 or int(g_ijs[j][i][s1]) == 1) and (int(g_ijs[i][j][s2]) == 1
239                                                   or int(g_ijs[j][i][s2]) == 1) and gamesPlayedInBetween == 0:
240           return 1
241       else:
242           return 0
243
244
245   def gameStatus(i: int, s: int, g_ijs: np.ndarray) -> str:
246       """
247       Gets the game status of a team.
```

```
248
249         Returns:
250         - HOME: plays a home game
251         - AWAY: plays an away game
252         - NIL : plays no game
253
254         As each team plays at most one game per time slot,
255         finding a positive result will exit the loop immediately.
256         """
257         teams_num = len(g_ijs)
258         result = "NIL"
259         for j in range(teams_num):
260             # Plays a home game?
261             if g_ijs[i][j][s] == 1:
262                 result = "HOME"
263                 break
264             # Plays an away game?
265             if g_ijs[j][i][s] == 1:
266                 result = "AWAY"
267                 break
268         return result
269
270
271     def hasBreak(i: int, s: int, g_ijs: np.ndarray) -> str:
272         """
273         Checks if the team i haa a break at slot s.
274
275         Returns:
276         - HOME for home break,
277         - AWAY for away break, and
278         - NIL  if there is no break
279         """
280         result = "NIL"
281         if s > 0:
282             currentstatus = gameStatus(i, s, g_ijs)
283             # If team i plays a (home or away) game
284             if currentstatus != "NIL":
285                 # then check all previous timeslots until you find a game that is not NIL
286                 s -= 1
287                 previousGameStatus = gameStatus(i, s, g_ijs)
288                 while s > 0 and previousGameStatus == "NIL":
289                     s -= 1
290                     previousGameStatus = gameStatus(i, s, g_ijs)
291                 # Now compare the current status with the previous status
292                 if currentstatus == previousGameStatus:
293                     result = currentstatus
294                 else:
295                     result = "NIL"
296         return result
297
298
299     # endregion
300
301     # region PARAMETERS, RESTRICTIONS & CONSTRAINTS
302
303     instance_and_parameters = {
304         "mode": "",                             # "canonical/feasible" for initializing a swarm of canonical/feasible
305                                                 # particles
306         "particles_num": 0,                     # Number of particles
307         "max_age": 0,                           # Maximum number of iterations (lifes)
308         "k": 0,                                 # Exponential coefficient in P = exp(-k * f_w / T)
309         "inertia_probability": 0,               # Probability of mutation (inertial coefficient)
310         "cognitive_probability": 0,             # Probability of crossover with pBest (cognitive coefficient)
311         "social_probability": 0,                # Probability of crossover with gBest (social coefficient)
312         "subparticle_fraction": 0,              # Number of violated hard constraints for generating sub-particles
```

```
313         "weights": {
314             "structural": 0,                    # Structural constraint weight
315             "hard": 0,                          # Hard constraint weight
316             "soft":{                            # Soft constraint weight per type
317                 "CA1": 0,
318                 "CA2": 0,
319                 "CA3": 0,
320                 "CA4": 0,
321                 "GA1": 0,
322                 "BR1": 0,
323                 "BR2": 0,
324                 "FA2": 0,
325                 "SE1": 0,
326             },
327         },
328 
329         "cpsat":{
330             "max_runtime": 0,                   # Maximum runtime of CP-SAT
331             "runtime_step": 0,                  # Runtime step of CP-SAT
332             "enhancing_mode": "",               # Mode for enhancing canonical particles.
333                                                 # Values: "HARD", "HARD_AS_SOFT", "ALL"
334             "static_period": 0,                 # Number of iterations with static
335         },
336 
337         "operators": {
338             "s_h_weight": 0,                    # swapHomes probability of occurence
339             "s_p_weight": 0,                    # swapOpponents probability of occurence
340             "s_t_weight": 0,                    # swapTeams probability of occurence
341             "s_r_weight": 0,                    # swapRounds probability of occurence
342             "s_m_weight": 0,                    # swapMatches probability of occurence
343             "r_t_weight": 0                     # rotateTeams probability of occurence
344         },
345 
346         "instanceName": "",                     # Instance name
347         "teams_num": 0,                         # Number of teams
348         "slots_num": 0,                         # Number of time slots
349         "RR": 0,                                # RR tournament format (NRR for RR = 0, kRR for RR != 0)
350         "compactness": "",                      # Compact or time-relaxed tournament
351         "gamemode": "",                         # Symmetry of the tournament
352         "objective": "",
353         "constraints": {}
354     }
355 
356 
357     def read_parameters_xml() -> None:
358         """ Read parameters XML file. """
359         # Parse the XML file
360         tree = et.parse('parameters.xml')
361         root = tree.getroot()
362 
363         # Extract PSO parameters
364         instance_and_parameters["mode"] = root.find('./PSO/Mode').text
365         instance_and_parameters["particles_num"] = int(root.find('./PSO/ParticlesNumber').text)
366         instance_and_parameters["max_age"] = int(root.find('./PSO/MaxAge').text)
367         instance_and_parameters["k"] = float(root.find('./PSO/ExponentialCoefficient').text)
368 
369         # Extract probabilities
370         instance_and_parameters["inertia_probability"] = float(root.find('./PSO/Probability/inertia').text)
371         instance_and_parameters["cognitive_probability"] = float(root.find('./PSO/Probability/cognitive').text)
372         instance_and_parameters["social_probability"] = float(root.find('./PSO/Probability/social').text)
373 
374         # Extract subparticles fraction
375         instance_and_parameters["subparticle_fraction"] = int(root.find('./PSO/Subparticles/fraction').text)
376 
377         # Extract weights
```

```
        instance_and_parameters["weights"]["structural"] = int(root.find('./PSO/Weights/structural').text)
        instance_and_parameters["weights"]["hard"] = int(root.find('./PSO/Weights/hard').text)
        instance_and_parameters["weights"]["soft"]["CA1"] = int(root.find('./PSO/Weights/soft/CA1').text)
        instance_and_parameters["weights"]["soft"]["CA2"] = int(root.find('./PSO/Weights/soft/CA2').text)
        instance_and_parameters["weights"]["soft"]["CA3"] = int(root.find('./PSO/Weights/soft/CA3').text)
        instance_and_parameters["weights"]["soft"]["CA4"] = int(root.find('./PSO/Weights/soft/CA4').text)
        instance_and_parameters["weights"]["soft"]["GA1"] = int(root.find('./PSO/Weights/soft/GA1').text)
        instance_and_parameters["weights"]["soft"]["BR1"] = int(root.find('./PSO/Weights/soft/BR1').text)
        instance_and_parameters["weights"]["soft"]["BR2"] = int(root.find('./PSO/Weights/soft/BR2').text)
        instance_and_parameters["weights"]["soft"]["FA2"] = int(root.find('./PSO/Weights/soft/FA2').text)
        instance_and_parameters["weights"]["soft"]["SE1"] = int(root.find('./PSO/Weights/soft/SE1').text)

        # Extract heuristic operator probabilities
        instance_and_parameters["operators"]["s_h_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   SwapHomes').text)
        instance_and_parameters["operators"]["s_p_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   SwapOpponents').text)
        instance_and_parameters["operators"]["s_t_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   SwapTeams').text)
        instance_and_parameters["operators"]["s_r_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   SwapRounds').text)
        instance_and_parameters["operators"]["s_m_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   SwapMatches').text)
        instance_and_parameters["operators"]["r_t_weight"] = float(root.find('./PSO/HeuristicOperators/Probability/
                                                                   RotateTeams').text)

        # Extract CPSAT parameters
        instance_and_parameters["cpsat"]["max_runtime"] = int(root.find('./CPSAT/MaxRuntime').text)
        instance_and_parameters["cpsat"]["runtime_step"] = int(root.find('./CPSAT/RuntimeStep').text)
        instance_and_parameters["cpsat"]["enhancing_mode"] = root.find('./CPSAT/EnhancingMode').text
        instance_and_parameters["cpsat"]["static_period"] = int(root.find('./CPSAT/StaticPeriod').text)


    def read_instance_xml(xmlFile: str) -> None:
        """ Read instance XML file. """
        tree = et.parse(xmlFile)
        root = tree.getroot()

        instance_and_parameters["instanceName"] = root.find("./MetaData/InstanceName").text

        teams = root.findall("./Resources/Teams/team")
        instance_and_parameters["teams_num"] = len(teams)

        slots = root.findall("./Resources/Slots/slot")
        instance_and_parameters["slots_num"] = len(slots)

        element = root.find("./Structure/Format/numberRoundRobin")
        instance_and_parameters["RR"] = 0 if element is None else int(element.text)

        element = root.find("./Structure/Format/compactness")
        instance_and_parameters["compactness"] = "" if element is None else element.text

        element = root.find("./Structure/Format/gameMode")
        instance_and_parameters["gamemode"] = "" if element is None else element.text

        element = root.find("./ObjectiveFunction/Objective")
        instance_and_parameters["objective"] = "" if element is None else element.text
        i = 0

        # Capacity Constraints
        for element in root.findall("./Constraints/CapacityConstraints/CA1"):
            instance_and_parameters["constraints"][f"c{i}"] = {}
            instance_and_parameters["constraints"][f"c{i}"]["category"] = "CA1"
            instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
                                                                for item in element.attrib["teams"].split(";")]
```

```
            instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
                                                    for item in element.attrib["slots"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
            instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
            instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["mode"]
            instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
            instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
            i += 1
        for element in root.findall("./Constraints/CapacityConstraints/CA2"):
            instance_and_parameters["constraints"][f"c{i}"] = {}
            instance_and_parameters["constraints"][f"c{i}"]["category"] = "CA2"
            instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
                                                    for item in element.attrib["teams1"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["teams2"] = [int(item)
                                                    for item in element.attrib["teams2"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
                                                    for item in element.attrib["slots"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
            instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
            instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["mode1"]
            instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode2"]
            instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
            instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
            i += 1
        for element in root.findall("./Constraints/CapacityConstraints/CA3"):
            instance_and_parameters["constraints"][f"c{i}"] = {}
            instance_and_parameters["constraints"][f"c{i}"]["category"] = "CA3"
            instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
                                                    for item in element.attrib["teams1"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["teams2"] = [int(item)
                                                    for item in element.attrib["teams2"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["k"] = int(element.attrib["intp"])
            instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
            instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
            instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["mode1"]
            instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode2"]
            instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
            instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
            i += 1
        for element in root.findall("./Constraints/CapacityConstraints/CA4"):
            instance_and_parameters["constraints"][f"c{i}"] = {}
            instance_and_parameters["constraints"][f"c{i}"]["category"] = "CA4"
            instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
                                                    for item in element.attrib["teams1"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["teams2"] = [int(item)
                                                    for item in element.attrib["teams2"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
                                                    for item in element.attrib["slots"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
            instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
            instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["mode1"]
            instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode2"]
            instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
            instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
            i += 1

        # Game Constraints
        for element in root.findall("./Constraints/GameConstraints/GA1"):
            instance_and_parameters["constraints"][f"c{i}"] = {}
            instance_and_parameters["constraints"][f"c{i}"]["category"] = "GA1"
            instance_and_parameters["constraints"][f"c{i}"]["meetings"] =
                                                     convert_str_to_meetings(element.attrib["meetings"])
            instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
                                                    for item in element.attrib["slots"].split(";")]
            instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
```

```
508             instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
509             instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
510             instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
511             i += 1
512
513         # Break Constraints
514         for element in root.findall("./Constraints/BreakConstraints/BR1"):
515             instance_and_parameters["constraints"][f"c{i}"] = {}
516             instance_and_parameters["constraints"][f"c{i}"]["category"] = "BR1"
517             instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
518                                                                          for item in element.attrib["teams"].split(";")]
519             instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
520                                                                            for item in element.attrib["slots"].split(";")]
521             instance_and_parameters["constraints"][f"c{i}"]["k"] = int(element.attrib["intp"])
522             instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["mode1"]
523             instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode2"]
524             instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
525             instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
526             i += 1
527         for element in root.findall("./Constraints/BreakConstraints/BR2"):
528             instance_and_parameters["constraints"][f"c{i}"] = {}
529             instance_and_parameters["constraints"][f"c{i}"]["category"] = "BR2"
530             instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
531                                                                          for item in element.attrib["teams"].split(";")]
532             instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
533                                                                            for item in element.attrib["slots"].split(";")]
534             instance_and_parameters["constraints"][f"c{i}"]["k"] = int(element.attrib["intp"])
535             instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib["homeMode"]
536             instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode2"]
537             instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
538             instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
539             i += 1
540
541         # Fairness Constraints
542         for element in root.findall("./Constraints/FairnessConstraints/FA2"):
543             instance_and_parameters["constraints"][f"c{i}"] = {}
544             instance_and_parameters["constraints"][f"c{i}"]["category"] = "FA2"
545             instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
546                                                                          for item in element.attrib["teams"].split(";")]
547             instance_and_parameters["constraints"][f"c{i}"]["timeslots"] = [int(item)
548                                                                            for item in element.attrib["slots"].split(";")]
549             instance_and_parameters["constraints"][f"c{i}"]["k"] = int(element.attrib["intp"])
550             instance_and_parameters["constraints"][f"c{i}"]["mode2"] = element.attrib["mode"]
551             instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
552             instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
553             i += 1
554
555         # Seperation Constraints
556         for element in root.findall("./Constraints/SeparationConstraints/SE1"):
557             instance_and_parameters["constraints"][f"c{i}"] = {}
558             instance_and_parameters["constraints"][f"c{i}"]["category"] = "SE1"
559             instance_and_parameters["constraints"][f"c{i}"]["teams1"] = [int(item)
560                                                                          for item in element.attrib["teams"].split(";")]
561             instance_and_parameters["constraints"][f"c{i}"]["k_min"] = int(element.attrib.get("min", 0))
562             instance_and_parameters["constraints"][f"c{i}"]["k_max"] = int(element.attrib.get("max", 0))
563             instance_and_parameters["constraints"][f"c{i}"]["mode1"] = element.attrib.get("mode1", "SLOTS")
564             instance_and_parameters["constraints"][f"c{i}"]["penalty"] = int(element.attrib["penalty"])
565             instance_and_parameters["constraints"][f"c{i}"]["ctype"] = element.attrib["type"]
566             i += 1
567
568
569     def get_instance_and_parameters(instance_xml_file: str) -> dict:
570         """
571         Read parameters from parameters.xml and instance from instance_xml_file
572
```

```
573         Arguments:
574         - XML file with instance data
575 
576         Returns:
577         - A dictionary
578         """
579         read_parameters_xml()
580         read_instance_xml(instance_xml_file)
581         # Apply compactness
582         instance_and_parameters["slots_num"] = compactness_restriction(instance_and_parameters)
583         return instance_and_parameters
584 
585 
586     def is_rule0_satisfied(g_ijs: np.ndarray) -> bool:
587         """ Check if each team satisfies Rule 0. """
588         teams_num: int = len(g_ijs)
589         slots_num: int = len(g_ijs[0][0])
590         for s in range(slots_num):
591             for i in range(teams_num):
592                 games: int = 0
593                 for j in range(teams_num):
594                     games += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
595                 if games > 1:
596                     return False
597         return True
598 
599 
600     def is_RR_satisfied(g_ijs: np.ndarray, RR: int, s1: int, s2: int) -> bool:
601         """ Check if Round-Robin Rule is satisfied. """
602         teams_num: int = len(g_ijs)
603         # Check kRR restriction
604         if RR > 0:
605             alreadyTested: list = []
606             for i in range(teams_num):
607                 for j in range(teams_num):
608                     # If i != j and (i,j) does not belong to the "alreadyTested" array
609                     if len(alreadyTested) == 0:
610                         doesNotBelong = True
611                     elif (i,j) in alreadyTested:
612                         doesNotBelong = False
613                     else:
614                         doesNotBelong = True
615 
616                     if (i != j) and doesNotBelong == True:
617                         g_ij = 0
618                         g_ji = 0
619                         for s in range(s1, s2):
620                             g_ij += int(g_ijs[i][j][s])
621                             g_ji += int(g_ijs[j][i][s])
622                         # Condition for kRR
623                         if (g_ij + g_ji != RR) or (abs(g_ij - g_ji) > 1):
624                             return False
625                         alreadyTested.append((i,j))
626                         alreadyTested.append((j,i))
627         return True
628 
629 
630     def compactness_restriction(instance_and_parameters: dict) -> int:
631         """
632         Check whether the tournament is compact or time-relaxed.
633 
634         If a tournament is compact, then the number of time slots
635         is calculated directly from RR and T, and any initial
636         value set in the parameter clause is bypassed.
637         """
```

```
638         slots_num: int = instance_and_parameters["slots_num"]
639         teams_num: int = instance_and_parameters["teams_num"]
640         rr: int = instance_and_parameters["RR"]
641         if instance_and_parameters["compactness"] == "C":
642             if rr > 0:
643                 slots_num = rr * (teams_num - 1) if teams_num % 2 == 0 else rr * teams_num
644         return slots_num
645 
646 
647     def is_phased(g_ijs: np.ndarray, instance_and_parameters: dict) -> bool:
648         """
649         Check if the timetable is phased.
650 
651         A timetable that satisfied kRR rule is divided into kRR phases.
652         Each phase is 1RR.
653         """
654         slots_num = instance_and_parameters["slots_num"]
655         phases_num = instance_and_parameters["RR"]
656         slots_per_phase = int(slots_num / phases_num)
657         flag: bool = True
658         for phase in range(phases_num):
659             flag = flag and is_RR_satisfied(g_ijs, 1, phase * slots_per_phase, (phase + 1) *slots_per_phase)
660         return flag
661 
662 
663     def CA1(teams: list, slots: list, k_min: int, k_max: int, mode: str, penalty: int, g_ijs: np.ndarray) -> int:
664         """
665         Implements the CA1 constraint.
666 
667         Arguments:
668         - teams: the teams of reference
669         - timeslots: the timeslots where the constraint is applied
670         - k_min: the minimum number of (home or away) games
671         - k_max: the maximum number of (home or away) games
672         - mode: home games or away games to be considered only
673         - penalty: this is the constraint weight
674 
675         Returns:
676         - total penalty
677         """
678         deviation: int = 0
679         teams_num: int = len(g_ijs)
680         for i in teams:
681             gamesNum = 0
682             # For home games
683             if mode == "H":
684                 for j in range(teams_num):
685                     for s in slots:
686                         gamesNum += int(g_ijs[i][j][s])
687             # For away games
688             elif mode == "A":
689                 for j in range(teams_num):
690                     for s in slots:
691                         gamesNum += int(g_ijs[j][i][s])
692             deviation += max([0, k_min - gamesNum, gamesNum - k_max])
693         return penalty * deviation
694 
695 
696     def CA2(teams1: list, teams2: list, slots, k_min: int, k_max: int, mode1: str, penalty: int, g_ijs: np.ndarray) -
697     > int:
698         """
699         Implements the CA2 constraint.
700 
701         Deviation di is calculated for each team in teams. Then, we sum up all the deviations.
702 
```

```
703         Arguments:
704         - teams1: the teams of reference
705         - teams2: the other teams for comparison
706         - timeslots: the timeslots where the constraint is applied
707         - k_min: the minimum number of (home or away or all) games
708         - k_max: the maximum number of (home or away or all) games
709         - mode1: home (H) games or away (A) games or all (HA) games
710         - mode2: always GLOBAL
711         - penalty: this is the constraint weight
712 
713         Returns:
714         - total penalty
715         """
716         deviation: int = 0
717         for i in teams1:
718             games_num: int = 0
719             # For home games
720             if mode1 == "H":
721                 for j in teams2:
722                     for s in slots:
723                         games_num += int(g_ijs[i][j][s])
724             # For away games
725             elif mode1 == "A":
726                 for j in teams2:
727                     for s in slots:
728                         games_num += int(g_ijs[j][i][s])
729             # For all games
730             elif mode1 == "HA":
731                 for j in teams2:
732                     for s in slots:
733                         games_num += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
734             deviation += max([0, k_min - games_num, games_num - k_max])
735         return penalty * deviation
736 
737 
738     def CA3(teams1: list, teams2: list, k: int, k_min: int, k_max: int, mode1: str, mode2: str, penalty: int, g_ijs:
739     np.ndarray) -> int:
740         """
741         Implements the CA3 constraint (for mode2 = SLOTS)
742 
743         Arguments:
744         - teams1: the teams of reference
745         - teams2: the other teams for comparison
746         - k: the number of time slots in a row
747         - k_min: the minimum number of (home or away) games
748         - k_max: the maximum number of (home or away) games
749         - mode1: home games or away games to be considered only
750         - mode2: slots or games
751 
752         Returns:
753         - total penalty
754         """
755         slots_num: int = len(g_ijs[0][0])
756         deviation: int = 0
757         for i in teams1:
758             # For home games
759             if mode1 == "H":
760                 for l in range(slots_num - k + 1):  # +1 added since the upper limit of range() is exclusive.
761                     innerSum: int = 0
762                     for j in teams2:
763                         for s in range(l, l + k):   # +1 added since the upper limit of range() is exclusive.
764                             innerSum += int(g_ijs[i][j][s])
765                     deviation += max([0, k_min - innerSum, innerSum - k_max])
766             # For away games
767             elif mode1 == "A":
```

```
768                 for l in range(slots_num - k + 1):
769                     innerSum: int = 0
770                     for j in teams2:
771                         for s in range(l, l + k):
772                             innerSum += int(g_ijs[j][i][s])
773                     deviation += max([0, k_min - innerSum, innerSum - k_max])
774             else:  # For HA games
775                 for l in range(slots_num - k + 1):
776                     innerSum: int = 0
777                     for j in teams2:
778                         for s in range(l, l + k):
779                             innerSum += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
780                     deviation += max([0, k_min - innerSum, innerSum - k_max])
781         return penalty * deviation
782 
783 
784     def CA4(teams1: list, teams2: list, slots: list, k_min: int, k_max: int, mode1: str, mode2: str, penalty: int,
785     g_ijs: np.ndarray) -> int:
786         """
787         Implements the CA4 constraint.
788 
789         Arguments:
790         - teams1: the teams of reference
791         - teams2: the other teams for comparison
792         - timeslots: the timeslots where the constraint is applied
793         - k_min: the minimum number of (home or away or all) games
794         - k_max: the maximum number of (home or away or all) games
795         - mode1: home (H) games or away (A) games or all (HA) games
796         - mode2: GLOBAL (for all teams in teams1 and teams2 and
797                  for all time slots in timeslots calculate a single deviation)
798                  EVERY (for each time slot in timeslots, calculate a deviation)
799         - penalty: this is the constraint weight
800 
801         Returns:
802         - total penalty
803         """
804         deviation: int = 0
805         # For all timeslots, calculate a single deviation
806         if mode2 == "GLOBAL":
807             games_num: int = 0
808             for i in teams1:
809                 for j in teams2:
810                     for s in slots:
811                         # For home games
812                         if mode1 == "H":
813                             games_num += int(g_ijs[i][j][s])
814                         # For away games
815                         elif mode1 == "A":
816                             games_num += int(g_ijs[j][i][s])
817                         # For all games
818                         elif mode1 == "HA":
819                             games_num += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
820             deviation = max([0, k_min - games_num, games_num - k_max])
821         # For each timeslot of the timeslots, calculate a deviation and sum them up
822         elif mode2 == "EVERY":
823             for s in slots:
824                 games_num: int = 0
825                 for i in teams1:
826                     for j in teams2:
827                          # For home games
828                         if mode1 == "H":
829                             games_num += int(g_ijs[i][j][s])
830                         # For away games
831                         elif mode1 == "A":
832                             games_num += int(g_ijs[j][i][s])
```

```
                    # For all games
                    elif mode1 == "HA":
                        games_num += int(g_ijs[i][j][s]) + int(g_ijs[j][i][s])
            deviation += max([0, k_min - games_num, games_num - k_max])
    return penalty * deviation


def GA1(meetings, slots: list, k_min: int, k_max: int, penalty: int, g_ijs: np.ndarray) -> int:
    """
    Implements the GA1 constraint.

    Arguments:
    - meetings: a set G of games (i,j)
    - timeslots: a set of time slots S
    - k_min, k_max: the minimum and maximun number of played games from the set G played
                    in the time slots of the set S respectively
    - penalty: this is the constraint weight

    Returns:
    - total penalty
    """
    games_num: int = 0
    for meeting in meetings:
        i: int = meeting[0]
        j: int = meeting[1]
        for s in slots:
            games_num += int(g_ijs[i][j][s])
    deviation: int = max([0, k_min - games_num, games_num - k_max])
    return penalty * deviation


def BR1(teams: list, slots: list, k: int, mode1: str, mode2: str, penalty: int, g_ijs: np.ndarray) -> int:
    """
    Implements the BR1 constraint.

    Arguments:
    - teams: the teams of reference
    - slots: the timeslots where the constraint is applied
    - k: the maximum number of (home or away or all) breaks per team i
    - mode1: home (H) games or away (A) games or all (HA) games
    - mode2: LEQ (each team in teams has at most k (mode1) breaks)
             " " (each team in teams has exactly k (mode1) breaks)
    - penalty: this is the constraint weight

    Returns:
    - the sum of deviation_i triggered by each team i
    """
    deviation: int = 0
    # For each team i count all mode1 breaks
    for i in teams:
        breaks_num: int = 0
        for s in slots:
            team_break: str = hasBreak(i, s, g_ijs)
            if mode2 == "H" and team_break == "HOME":
                breaks_num += 1
            elif mode2 == "A" and team_break == "AWAY":
                breaks_num += 1
            elif mode2 == "HA" and team_break != "NIL":
                breaks_num += 1
        # Calculate deviation for team i
        if mode1 == "LEQ":        # Each team in teams has at most k (mode2) breaks
            if breaks_num > k:
                deviation += breaks_num - k
        else:                     # Each team in teams has exactly k (mode2) breaks
            if breaks_num != k:
```

```
                deviation += abs(breaks_num - k)
    return penalty * deviation


def BR2(teams: list, slots: list, k: int, mode1: str, mode2: str, penalty: int, g_ijs: np.ndarray) -> int:
    """
    Implements the BR2 constraint.

    Arguments:
    - teams: the teams of reference
    - timeslots: the timeslots where the constraint is applied
    - k: the maximum number of (homeMode) breaks for all teams of the set teams
    - mode1: also named as homeMode: home (H) games or away (A) games or all (HA) games
    - mode2: LEQ (each team in teams has at most k (mode2) breaks)
                "" (each team in teams has exactly k (mode2) breaks)
    - penalty: this is the constraint weight

    Returns:
    - the deviation for the whole set of teams.
    """
    deviation: int = 0
    breaks_num: int = 0
    # For all teams i count all mode1 breaks
    for i in teams:
        for s in slots:
            team_break: str = hasBreak(i, s, g_ijs)
            if mode1 == "H" and team_break == "HOME":
                breaks_num += 1
            elif mode1 == "A" and team_break == "AWAY":
                breaks_num += 1
            elif mode1 == "HA" and team_break != "NIL":
                breaks_num += 1
    # Calculate deviation for total number of breaks found
    if mode2 == "LEQ":             # Total (homeMode) breaks are at most k
        if breaks_num > k:
            deviation = breaks_num - k
    else:                          # Total (homeMode) breaks are exactly k
        if breaks_num != k:
            deviation = abs(breaks_num - k)
    return penalty * deviation


def FA2(teams: list, slots: list, k: int, mode2: str, penalty: int, g_ijs: np.ndarray) -> int:
    """
    Implements the FA2 constraint.

    Arguments:
    - teams: the teams of reference
    - timeslots: the timeslots where the constraint is applied
    - k: the maximum number of the sum of home game differences between team i and team j
    - mode2: H for home games
    - penalty: this is the constraint weight

    Returns:
    - the total penalty
    """
    deviation: int = 0
    if mode2 == "H":
        for i in teams:
            for j in teams:
                if i < j:
                    a = [0]
                    for s in slots:
                        a.append(abs(sum(g_ijs[i][t][p] for t in teams for p in range(s + 1)) -
                                     sum(g_ijs[j][t][p] for t in teams for p in range(s + 1))) - k)
```

```
963                         deviation += max(a)
964         return penalty * deviation
965 
966 
967     def SE1(teams: list, k_min: int, k_max: int, mode1: str, penalty: int, g_ijs: np.ndarray) -> int:
968         """
969         Implements the SE1 constraint.
970 
971         Arguments:
972         - teams: the teams of reference
973         - k_min, k_max are the low and top limit of time slots in between two consecutive games of i,j
974         - mode1 = SLOTS
975         - penalty: this is the constraint weight
976 
977         Returns:
978         - the total penalty
979         """
980         slots_num: int = len(g_ijs[0][0])
981         deviation: int = 0
982         if mode1 == "SLOTS":
983             for i in teams:
984                 for j in teams:
985                     if i < j:
986                         for s1 in range(slots_num):
987                             for s2 in range(s1 + 1, slots_num):
988                                 if k_max == 0:
989                                     a = [0, k_min - (s2 - s1 - 1)]
990                                 else:
991                                     a = [0, k_min - (s2 - s1 - 1), (s2 - s1 - 1) - k_max]
992                                 deviation += y_function(i, j, s1, s2, g_ijs) * max(a)
993         return penalty * deviation
994 
995 
996     def get_constraint_cost(key: str, instance_and_parameters: dict, g_ijs: np.ndarray) -> tuple[int, int]:
997         """
998         Get the constraint fitness.
999 
1000        Weighted cost is only for soft constraints.
1001        Hard constraints have a global weight.
1002 
1003        Arguments:
1004        - constraint key: string
1005        - problem parameters: dictionary
1006        - timetable in g encoding: np.ndarray
1007 
1008        Returns:
1009        - real code, weighted cost
1010        """
1011        real_cost: int = 0
1012        match instance_and_parameters["constraints"][key]["category"]:
1013            case "CA1":
1014                teams: list = instance_and_parameters["constraints"][key]["teams1"]
1015                slots = instance_and_parameters["constraints"][key]["timeslots"]
1016                k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
1017                k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
1018                mode: str = instance_and_parameters["constraints"][key]["mode1"]
1019                penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
1020                real_cost: int = CA1(teams, slots, k_min, k_max, mode, penalty, g_ijs)
1021                weighted_cost: int = instance_and_parameters["weights"]["soft"]["CA1"] * real_cost
1022            case "CA2":
1023                teams1: list = instance_and_parameters["constraints"][key]["teams1"]
1024                teams2: list = instance_and_parameters["constraints"][key]["teams2"]
1025                slots: list = instance_and_parameters["constraints"][key]["timeslots"]
1026                k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
1027                k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
```

```
            mode: str = instance_and_parameters["constraints"][key]["mode1"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = CA2(teams1, teams2, slots, k_min, k_max, mode, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["CA2"] * real_cost
        case "CA3":
            teams1: list = instance_and_parameters["constraints"][key]["teams1"]
            teams2: list = instance_and_parameters["constraints"][key]["teams2"]
            k = int(instance_and_parameters["constraints"][key]["k"])
            k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
            k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
            mode1: str = instance_and_parameters["constraints"][key]["mode1"]
            mode2: str = instance_and_parameters["constraints"][key]["mode2"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = CA3(teams1, teams2, k, k_min, k_max, mode1, mode2, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["CA3"] * real_cost
        case "CA4":
            teams1: list = instance_and_parameters["constraints"][key]["teams1"]
            teams2: list = instance_and_parameters["constraints"][key]["teams2"]
            slots: list = instance_and_parameters["constraints"][key]["timeslots"]
            k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
            k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
            mode1: str = instance_and_parameters["constraints"][key]["mode1"]
            mode2: str = instance_and_parameters["constraints"][key]["mode2"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = CA4(teams1, teams2, slots, k_min, k_max, mode1, mode2, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["CA4"] * real_cost
        case "GA1":
            meetings: list = instance_and_parameters["constraints"][key]["meetings"]
            slots: list = instance_and_parameters["constraints"][key]["timeslots"]
            k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
            k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = GA1(meetings, slots, k_min, k_max, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["GA1"] * real_cost
        case "BR1":
            teams: list = instance_and_parameters["constraints"][key]["teams1"]
            slots: list = instance_and_parameters["constraints"][key]["timeslots"]
            k: int = int(instance_and_parameters["constraints"][key]["k"])
            mode1: str = instance_and_parameters["constraints"][key]["mode1"]
            mode2: str = instance_and_parameters["constraints"][key]["mode2"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = BR1(teams, slots, k, mode1, mode2, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["BR1"] * real_cost
        case "BR2":
            teams: list = instance_and_parameters["constraints"][key]["teams1"]
            slots: list = instance_and_parameters["constraints"][key]["timeslots"]
            k: int = int(instance_and_parameters["constraints"][key]["k"])
            mode1: str = instance_and_parameters["constraints"][key]["mode1"]
            mode2: str = instance_and_parameters["constraints"][key]["mode2"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = BR2(teams, slots, k, mode1, mode2, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["BR2"] * real_cost
        case "FA2":
            teams: list = instance_and_parameters["constraints"][key]["teams1"]
            slots: list = instance_and_parameters["constraints"][key]["timeslots"]
            k: int = int(instance_and_parameters["constraints"][key]["k"])
            mode2: str = instance_and_parameters["constraints"][key]["mode2"]
            penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
            real_cost: int = FA2(teams, slots, k, mode2, penalty, g_ijs)
            weighted_cost: int = instance_and_parameters["weights"]["soft"]["FA2"] * real_cost
        case "SE1":
            teams: list = instance_and_parameters["constraints"][key]["teams1"]
            k_min: int = int(instance_and_parameters["constraints"][key]["k_min"])
            k_max: int = int(instance_and_parameters["constraints"][key]["k_max"])
            mode1: str = instance_and_parameters["constraints"][key]["mode1"]
```

```
1093                penalty: int = int(instance_and_parameters["constraints"][key]["penalty"])
1094                real_cost: int = SE1(teams, k_min, k_max, mode1, penalty, g_ijs)
1095                weighted_cost: int = instance_and_parameters["weights"]["soft"]["SE1"] * real_cost
1096            case _:
1097                pass
1098        return real_cost, weighted_cost
1099
1100    # endregion
```

Πίνακας 26: Βοηθητικές συναρτήσεις

Το πρόγραμμα διαβάζει επίσης τις παραμέτρους του μοντέλου από ένα αρχείο XML με όνομα parameters.xml που φαίνεται στον ακόλουθο πίνακα:

```
1   <?xml version='1.0' encoding='utf-8'?>
2   <Parameters>
3       <PSO>
4           <Mode>canonical</Mode>
5           <ParticlesNumber>4</ParticlesNumber>
6           <MaxAge>10000</MaxAge>
7           <ExponentialCoefficient>6.7</ExponentialCoefficient>
8           <Probability>
9               <inertia>0.1969</inertia>
10              <cognitive>0.4015</cognitive>
11              <social>0.4015</social>
12          </Probability>
13          <Subparticles>
14              <fraction>1</fraction>
15          </Subparticles>
16          <Weights>
17              <structural>100000</structural>
18              <hard>10</hard>
19              <soft>
20                  <CA1>1</CA1>
21                  <CA2>1</CA2>
22                  <CA3>1</CA3>
23                  <CA4>1</CA4>
24                  <GA1>1</GA1>
25                  <BR1>1</BR1>
26                  <BR2>1</BR2>
27                  <FA2>1</FA2>
28                  <SE1>1</SE1>
29              </soft>
30          </Weights>
31          <HeuristicOperators>
32              <Probability>
33                  <SwapHomes>0.20</SwapHomes>
34                  <SwapOpponents>0.23</SwapOpponents>
35                  <SwapTeams>0.15</SwapTeams>
36                  <SwapRounds>0.14</SwapRounds>
37                  <SwapMatches>0.16</SwapMatches>
38                  <RotateTeams>0.12</RotateTeams>
39              </Probability>
40          </HeuristicOperators>
41      </PSO>
42      <CPSAT>
43          <MaxRuntime>10</MaxRuntime>
```

```
44            <RuntimeStep>0</RuntimeStep>
45            <EnhancingMode>HARD_AS_SOFT</EnhancingMode>
46            <StaticPeriod>30</StaticPeriod>
47        </CPSAT>
48    </Parameters>
```

Πίνακας 27: Αρχείο parameters.xml

□

Παράρτημα Β

Στο παράρτημα αυτό παραθέτουμε αναλυτικά όλες τις λύσεις που βρήκαμε επιλύοντας τα προβλήματα του διαγωνισμού ITC2021 με την βοήθεια του αλγορίθμου CP-PSO:

Πρόβλημα Test 1

Η λύση που βρήκαμε για το πρόβλημα Test 1 είναι η (0, 1066) που έχει ανάλυση κόστους:

CA1: 7
CA3: 155
GA1: 4
SE1: 900

και ωρολόγιο πρόγραμμα:

slot 0	(1,0)	(3,2)	(5,4)
slot 1	(1,3)	(2,5)	(4,0)
slot 2	(0,5)	(2,1)	(3,4)
slot 3	(0,2)	(4,1)	(5,3)
slot 4	(1,5)	(3,0)	(4,2)
slot 5	(0,4)	(2,3)	(5,1)
slot 6	(1,4)	(2,0)	(3,5)
slot 7	(0,1)	(4,3)	(5,2)
slot 8	(0,3)	(1,2)	(4,5)
slot 9	(2,4)	(3,1)	(5,0)

Πρόβλημα Test 2

Η λύση που βρήκαμε για το πρόβλημα Test 2 είναι η (0, 176) με ανάλυση κόστους:

CA1: 11
CA2: 165

και ωρολόγιο πρόγραμμα:

slot 0	(1,5)	(2,0)	(4,3)
slot 1	(2,1)	(3,0)	(5,4)
slot 2	(0,1)	(2,3)	(4,5)

slot 3 (1,4) (3,2) (5,0)
slot 4 (0,2) (4,1) (5,3)
slot 5 (0,4) (2,5) (3,1)
slot 6 (1,3) (4,0) (5,2)
slot 7 (0,5) (1,2) (3,4)
slot 8 (1,0) (2,4) (3,5)
slot 9 (0,3) (4,2) (5,1)

Η λύση που βρήκαμε έχει την ίδια τιμή αντικειμενικής συνάρτησης με την βέλτιστη λύση που είναι δημοσιευμένη στην ιστοσελίδα του διαγωνισμού ITC2021, αλλά το δικό μας ωρολόγιο πρόγραμμα είναι **διαφορετικό**. Χρησιμοποιώντας την σχέση (55) βρίσκουμε ότι η απόσταση d ανάμεσα στην δική μας λύση και την λύση του διαγωνισμού ITC2021 ισούται με $d = 0.7333$ γεγονός που σημαίνει ότι πρόκειται για μία πολύ διαφορετική λύση που επιτυγχάνει όμως το ίδιο κόστος.

Πρόβλημα Test 3

Η λύση που βρήκαμε για το πρόβλημα Test 3 είναι η (0, 1253) με ανάλυση κόστους:

CA1: 18
CA3: 485
CA4: 750

και ωρολόγιο πρόγραμμα:

slot 0 (0,2) (1,4) (5,3)
slot 1 (2,5) (3,1) (4,0)
slot 2 (1,0) (3,2) (4,5)
slot 3 (0,3) (2,4) (5,1)
slot 4 (1,2) (4,3) (5,0)
slot 5 (2,0) (3,5) (4,1)
slot 6 (0,4) (1,3) (5,2)
slot 7 (1,5) (3,0) (4,2)
slot 8 (0,1) (2,3) (5,4)
slot 9 (0,5) (2,1) (3,4)

Πρόβλημα Test 4

Η λύση που βρήκαμε για το πρόβλημα Test 4 είναι η (0, 4533) με ανάλυση κόστους:

CA1: 21
CA2: 905
CA3: 830
CA4: 1725
GA1: 4
BR1: 10
BR2: 140
SE1: 900

και ωρολόγιο πρόγραμμα:

slot 0 (2,0) (3,5) (4,1)
slot 1 (0,4) (3,1) (5,2)
slot 2 (0,3) (1,5) (2,4)
slot 3 (1,2) (4,3) (5,0)
slot 4 (0,1) (3,2) (5,4)
slot 5 (1,4) (2,5) (3,0)
slot 6 (2,1) (4,0) (5,3)
slot 7 (0,2) (3,4) (5,1)
slot 8 (1,0) (2,3) (4,5)
slot 9 (0,5) (1,3) (4,2)

Πρόβλημα Test 5

Η λύση που βρήκαμε για το πρόβλημα Test 5 είναι η (0, 2) με ανάλυση κόστους:

CA1: 2

και ωρολόγιο πρόγραμμα:

slot 0 (2,12) (3,7) (5,10) (6,1) (8,4) (9,13) (11,0) (15,14)
slot 1 (0,12) (4,7) (5,2) (6,8) (9,1) (10,15) (11,14) (13,3)
slot 2 (1,3) (2,0) (4,11) (5,7) (9,12) (10,6) (13,14) (15,8)
slot 3 (1,4) (2,7) (5,13) (6,9) (10,11) (12,8) (14,3) (15,0)
slot 4 (2,10) (3,9) (4,0) (5,1) (7,11) (8,13) (12,14) (15,6)
slot 5 (4,3) (6,2) (8,5) (10,9) (12,1) (13,7) (14,0) (15,11)
slot 6 (0,3) (1,11) (5,6) (8,9) (10,4) (13,12) (14,7) (15,2)

slot 7 (0,6) (1,7) (2,9) (3,11) (4,14) (8,10) (12,5) (15,13)
slot 8 (0,9) (1,14) (2,4) (3,8) (6,13) (7,15) (10,12) (11,5)
slot 9 (0,10) (2,13) (4,6) (8,11) (9,7) (12,3) (14,5) (15,1)
slot 10 (1,10) (3,5) (4,9) (6,14) (7,0) (8,2) (11,13) (15,12)
slot 11 (5,0) (8,1) (10,7) (11,2) (12,6) (13,4) (14,9) (15,3)
slot 12 (1,0) (2,3) (4,12) (6,7) (8,14) (9,11) (13,10) (15,5)
slot 13 (0,8) (3,10) (5,9) (6,11) (7,12) (13,1) (14,2) (15,4)
slot 14 (2,1) (3,6) (5,4) (8,7) (10,14) (11,12) (13,0) (15,9)
slot 15 (0,2) (1,5) (4,8) (7,13) (9,3) (11,10) (12,15) (14,6)
slot 16 (0,7) (1,13) (3,4) (5,12) (9,2) (10,8) (11,6) (14,15)
slot 17 (0,15) (1,8) (2,11) (3,12) (4,5) (6,10) (7,9) (14,13)
slot 18 (0,1) (2,6) (4,10) (5,14) (7,3) (11,8) (12,9) (13,15)
slot 19 (1,6) (3,0) (7,4) (8,15) (9,5) (12,2) (13,11) (14,10)
slot 20 (0,14) (6,3) (8,12) (9,4) (10,2) (11,1) (13,5) (15,7)
slot 21 (2,14) (3,13) (5,15) (7,1) (8,6) (9,10) (11,4) (12,0)
slot 22 (0,4) (6,12) (7,5) (8,3) (9,15) (10,1) (13,2) (14,11)
slot 23 (1,15) (2,8) (7,14) (9,0) (10,5) (11,3) (12,4) (13,6)
slot 24 (2,15) (3,1) (4,13) (6,5) (8,0) (9,14) (11,7) (12,10)
slot 25 (0,11) (1,12) (2,5) (3,15) (7,10) (9,6) (13,8) (14,4)
slot 26 (3,14) (4,1) (5,11) (6,15) (7,2) (9,8) (10,0) (12,13)
slot 27 (0,13) (1,2) (4,15) (5,8) (7,6) (10,3) (11,9) (14,12)
slot 28 (0,5) (1,9) (3,2) (6,4) (10,13) (11,15) (12,7) (14,8)
slot 29 (4,2) (5,3) (6,0) (7,8) (12,11) (13,9) (14,1) (15,10)

Η παραπάνω λύση έχει την ίδια τιμή αντικειμενικής συνάρτησης με την βέλτιστη λύση που είναι δημοσιευμένη από τον διαγωνισμό ITC2021, αλλά **διαφορετικό** ωρολόγιο πρόγραμμα. Η απόσταση d (σχέση 55) ανάμεσα στην δική μας λύση και την λύση του διαγωνισμού ισούται με $d = 0.8667$, γεγονός που σημαίνει ότι πρόκειται για μία πολύ διαφορετική λύση.

Πρόβλημα Early 1

Η λύση που βρήκαμε είναι η (0, 1603) με ανάλυση κόστους

CA1: 6

CA4: 690

GA1: 7

SE1: 900

και ωρολόγιο πρόγραμμα:

slot 0 (3,0) (6,5) (7,9) (8,2) (10,13) (11,1) (12,14) (15,4)
slot 1 (0,7) (1,6) (2,12) (3,8) (4,11) (5,15) (9,10) (14,13)
slot 2 (1,7) (6,3) (8,4) (10,2) (11,12) (13,9) (14,5) (15,0)
slot 3 (0,5) (2,13) (3,10) (4,1) (7,14) (8,15) (9,11) (12,6)
slot 4 (1,2) (5,7) (6,4) (10,12) (11,0) (13,8) (14,3) (15,9)
slot 5 (0,14) (2,6) (3,9) (4,5) (7,10) (8,11) (12,1) (15,13)
slot 6 (0,2) (5,3) (6,8) (9,12) (10,4) (11,7) (13,1) (14,15)
slot 7 (1,9) (2,5) (3,12) (6,0) (7,4) (8,14) (13,11) (15,10)
slot 8 (0,9) (4,3) (5,13) (7,15) (10,1) (11,6) (12,8) (14,2)
slot 9 (2,11) (3,7) (5,12) (6,14) (8,10) (9,4) (13,0) (15,1)
slot 10 (1,8) (4,0) (7,2) (9,6) (10,5) (11,14) (12,13) (15,3)
slot 11 (0,1) (2,4) (5,11) (6,7) (8,9) (12,15) (13,3) (14,10)
slot 12 (0,12) (1,5) (3,2) (4,13) (7,8) (9,14) (11,10) (15,6)
slot 13 (2,15) (3,11) (5,9) (8,0) (10,6) (12,4) (13,7) (14,1)
slot 14 (0,10) (1,3) (4,14) (5,8) (6,13) (9,2) (11,15) (12,7)
slot 15 (0,3) (2,9) (4,8) (6,11) (7,1) (10,15) (13,5) (14,12)
slot 16 (1,0) (3,15) (4,6) (5,14) (8,7) (9,13) (11,2) (12,10)
slot 17 (0,4) (2,1) (7,6) (9,3) (10,8) (13,12) (14,11) (15,5)
slot 18 (1,15) (3,5) (4,7) (6,2) (8,13) (10,9) (12,11) (14,0)
slot 19 (1,14) (2,10) (5,6) (7,3) (9,0) (11,8) (13,4) (15,12)
slot 20 (0,11) (3,13) (4,2) (6,9) (8,1) (10,7) (12,5) (15,14)
slot 21 (1,12) (2,7) (3,6) (5,0) (9,15) (11,4) (13,10) (14,8)
slot 22 (1,13) (2,0) (4,9) (7,5) (8,6) (10,14) (12,3) (15,11)
slot 23 (0,8) (3,4) (5,1) (6,10) (11,9) (12,2) (13,15) (14,7)
slot 24 (1,11) (2,3) (4,15) (7,13) (8,12) (9,5) (10,0) (14,6)
slot 25 (0,13) (2,8) (3,14) (5,4) (6,12) (9,1) (10,11) (15,7)
slot 26 (1,10) (6,15) (7,0) (8,3) (11,5) (12,9) (13,2) (14,4)
slot 27 (0,6) (2,14) (3,1) (4,12) (5,10) (9,7) (11,13) (15,8)
slot 28 (1,4) (7,11) (8,5) (10,3) (12,0) (13,6) (14,9) (15,2)
slot 29 (0,15) (4,10) (5,2) (6,1) (7,12) (9,8) (11,3) (13,14)

Πρόβλημα Early 2

Για το πρόβλημα αυτό η λύση που βρήκαμε είναι η (0, 713) που έχει ανάλυση κόστους

CA1: 28

CA3: 645

FA2: 40

και ωρολόγιο πρόγραμμα:

slot 0 (0,3) (4,11) (5,8) (7,1) (12,10) (13,2) (14,9) (15,6)
slot 1 (1,13) (2,9) (3,5) (6,8) (10,4) (11,7) (12,0) (14,15)
slot 2 (0,2) (1,11) (3,10) (4,12) (8,14) (9,7) (13,6) (15,5)
slot 3 (2,6) (4,8) (5,9) (7,13) (10,1) (11,15) (12,3) (14,0)
slot 4 (0,13) (1,5) (2,15) (3,14) (6,4) (8,10) (9,11) (12,7)
slot 5 (4,3) (5,12) (7,6) (9,1) (11,0) (13,10) (14,2) (15,8)
slot 6 (0,5) (1,4) (3,11) (6,14) (7,2) (8,13) (10,15) (12,9)
slot 7 (2,3) (5,11) (6,10) (8,7) (9,0) (13,12) (14,1) (15,4)
slot 8 (0,8) (1,2) (3,6) (4,9) (7,14) (10,5) (11,13) (12,15)
slot 9 (5,7) (6,0) (8,2) (9,3) (12,11) (13,4) (14,10) (15,1)
slot 10 (1,12) (2,5) (4,14) (7,15) (9,6) (10,0) (11,8) (13,3)
slot 11 (0,1) (2,10) (3,7) (5,4) (6,11) (8,12) (14,13) (15,9)
slot 12 (1,8) (4,0) (5,14) (9,13) (10,7) (11,2) (12,6) (15,3)
slot 13 (0,15) (2,12) (3,8) (6,1) (7,4) (10,9) (13,5) (14,11)
slot 14 (0,7) (1,3) (4,2) (5,6) (8,9) (11,10) (12,14) (15,13)
slot 15 (2,4) (3,1) (6,15) (7,5) (9,8) (10,11) (13,0) (14,12)
slot 16 (0,6) (1,15) (2,14) (4,13) (5,3) (7,10) (11,9) (12,8)
slot 17 (1,0) (3,12) (6,7) (8,4) (9,5) (10,2) (13,14) (15,11)
slot 18 (2,13) (4,6) (5,1) (7,0) (8,15) (9,10) (11,12) (14,3)
slot 19 (0,11) (1,9) (3,4) (6,5) (7,8) (10,13) (12,2) (15,14)
slot 20 (2,0) (4,5) (8,3) (9,12) (11,1) (13,7) (14,6) (15,10)
slot 21 (1,14) (3,15) (5,0) (6,13) (7,11) (9,2) (10,8) (12,4)
slot 22 (0,12) (2,1) (4,10) (8,6) (11,3) (13,9) (14,5) (15,7)
slot 23 (0,4) (2,7) (3,13) (5,15) (8,11) (9,14) (10,6) (12,1)
slot 24 (1,10) (3,0) (6,9) (7,12) (11,5) (13,8) (14,4) (15,2)
slot 25 (0,14) (2,11) (4,7) (6,3) (8,5) (9,15) (10,12) (13,1)
slot 26 (0,9) (1,7) (3,2) (4,15) (5,10) (11,6) (12,13) (14,8)
slot 27 (5,2) (6,12) (7,3) (8,1) (9,4) (10,14) (13,11) (15,0)
slot 28 (0,10) (1,6) (2,8) (3,9) (11,4) (12,5) (13,15) (14,7)
slot 29 (4,1) (5,13) (6,2) (7,9) (8,0) (10,3) (11,14) (15,12)

Πρόβλημα Early 3

Για το πρόβλημα αυτό βρήκαμε την λύση (0, 1419) που έχει ανάλυση κόστους:

CA2: 80

CA3: 480

GA1: 19

BR2: 840

και ωρολόγιο πρόγραμμα:

slot 0 (1,10) (4,8) (5,0) (7,15) (9,3) (12,11) (13,6) (14,2)
slot 1 (0,13) (2,5) (3,7) (6,1) (8,9) (11,14) (12,4) (15,10)
slot 2 (1,0) (2,7) (5,3) (8,11) (9,15) (10,12) (13,4) (14,6)
slot 3 (0,9) (3,10) (4,14) (5,12) (6,8) (7,1) (11,2) (15,13)
slot 4 (1,15) (2,4) (6,11) (7,0) (8,5) (10,9) (13,12) (14,3)
slot 5 (0,10) (1,9) (4,6) (5,7) (11,15) (12,3) (13,2) (14,8)
slot 6 (0,14) (2,1) (3,4) (5,11) (7,12) (9,13) (10,6) (15,8)
slot 7 (3,2) (4,5) (6,15) (8,1) (9,11) (12,0) (13,7) (14,10)
slot 8 (0,6) (1,14) (5,13) (8,2) (9,12) (10,7) (11,4) (15,3)
slot 9 (2,0) (3,8) (6,9) (7,4) (10,11) (12,15) (13,1) (14,5)
slot 10 (0,15) (1,3) (4,10) (6,5) (8,13) (9,14) (11,7) (12,2)
slot 11 (3,13) (4,0) (5,9) (7,6) (10,2) (11,1) (12,8) (15,14)
slot 12 (0,3) (1,12) (6,2) (8,10) (9,4) (13,11) (14,7) (15,5)
slot 13 (1,5) (2,9) (3,6) (4,15) (7,8) (10,13) (11,0) (12,14)
slot 14 (0,8) (4,1) (5,10) (6,12) (9,7) (11,3) (13,14) (15,2)
slot 15 (2,13) (3,15) (7,5) (8,0) (10,1) (11,6) (12,9) (14,4)
slot 16 (0,5) (2,12) (4,7) (6,14) (9,10) (11,8) (13,3) (15,1)
slot 17 (1,11) (3,0) (5,2) (6,4) (7,10) (8,12) (14,13) (15,9)
slot 18 (0,7) (2,11) (3,1) (5,15) (9,6) (10,4) (13,8) (14,12)
slot 19 (1,2) (6,3) (7,9) (8,14) (11,10) (12,5) (13,0) (15,4)
slot 20 (1,7) (2,8) (3,12) (4,11) (5,6) (10,15) (13,9) (14,0)
slot 21 (0,11) (1,4) (3,14) (5,8) (7,13) (9,2) (12,10) (15,6)
slot 22 (0,1) (2,15) (4,3) (6,13) (8,7) (10,5) (11,12) (14,9)
slot 23 (1,6) (2,3) (5,4) (7,14) (9,8) (10,0) (12,13) (15,11)
slot 24 (0,12) (3,11) (4,2) (6,7) (8,15) (9,5) (13,10) (14,1)
slot 25 (2,14) (4,9) (5,1) (7,3) (10,8) (11,13) (12,6) (15,0)
slot 26 (0,4) (2,10) (3,5) (8,6) (9,1) (12,7) (13,15) (14,11)
slot 27 (1,8) (4,13) (5,14) (6,0) (7,2) (10,3) (11,9) (15,12)
slot 28 (0,2) (3,9) (6,10) (7,11) (8,4) (12,1) (13,5) (14,15)
slot 29 (1,13) (2,6) (4,12) (8,3) (9,0) (10,14) (11,5) (15,7)

Πρόβλημα Early 4

Το αποτέλεσμα που βρήκαμε είναι (9, 2147) το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους:

Ισχυροί περιορισμοί

164 = 155 ικανοποιούνται + 9 παραβιάζονται

CA4: 4

GA1: 4

BR1: 1

Ασθενείς περιορισμοί

CA1: 27

CA2: 800

SE1: 1320

και ωρολόγιο πρόγραμμα:

slot 0 (1,15) (2,3) (4,0) (9,17) (10,6) (11,8) (12,16) (13,7) (14,5)
slot 1 (3,9) (5,13) (6,11) (7,2) (8,0) (12,1) (15,4) (16,10) (17,14)
slot 2 (0,6) (1,9) (2,12) (4,5) (7,16) (8,15) (10,14) (11,3) (17,13)
slot 3 (2,4) (3,7) (5,11) (10,9) (12,0) (13,1) (14,8) (15,6) (16,17)
slot 4 (0,10) (1,7) (4,17) (5,15) (6,16) (8,3) (9,12) (11,13) (14,2)
slot 5 (2,8) (3,6) (7,0) (10,1) (12,4) (13,9) (15,11) (16,14) (17,5)
slot 6 (0,15) (1,14) (3,16) (4,13) (5,2) (6,12) (8,10) (9,7) (11,17)
slot 7 (5,1) (7,6) (11,2) (12,10) (13,3) (14,0) (15,9) (16,4) (17,8)
slot 8 (0,16) (1,17) (2,6) (3,12) (4,11) (5,7) (9,14) (10,15) (13,8)
slot 9 (0,2) (6,17) (7,4) (8,1) (11,10) (12,5) (14,3) (15,13) (16,9)
slot 10 (1,6) (2,15) (3,0) (8,4) (9,5) (13,10) (14,7) (16,11) (17,12)
slot 11 (0,1) (2,9) (3,17) (4,10) (5,16) (6,13) (7,8) (11,14) (15,12)
slot 12 (1,4) (8,5) (9,6) (10,3) (12,11) (13,0) (15,14) (16,2) (17,7)
slot 13 (0,9) (2,17) (4,3) (5,10) (6,8) (7,12) (11,1) (14,13) (16,15)
slot 14 (0,11) (1,2) (5,3) (8,12) (9,4) (10,7) (13,16) (14,6) (15,17)
slot 15 (2,10) (3,1) (4,14) (6,5) (7,15) (9,11) (12,13) (16,8) (17,0)
slot 16 (0,5) (1,16) (4,6) (8,9) (10,17) (11,7) (13,2) (14,12) (15,3)
slot 17 (2,16) (5,4) (6,1) (7,3) (9,13) (11,0) (12,8) (14,10) (17,15)
slot 18 (3,11) (4,1) (7,17) (8,6) (10,2) (12,9) (13,14) (15,5) (16,0)
slot 19 (0,7) (1,10) (2,13) (5,12) (6,15) (9,8) (11,4) (14,16) (17,3)
slot 20 (3,5) (4,2) (6,0) (7,13) (8,17) (10,16) (11,9) (12,14) (15,1)
slot 21 (0,3) (2,11) (5,8) (7,10) (9,15) (13,12) (14,4) (16,6) (17,1)
slot 22 (1,0) (3,13) (4,9) (6,7) (8,14) (10,5) (11,12) (15,16) (17,2)

slot 23 (0,8) (2,7) (5,17) (6,10) (9,1) (12,15) (13,4) (14,11) (16,3)
slot 24 (1,12) (3,10) (4,16) (7,14) (8,11) (9,2) (13,5) (15,0) (17,6)
slot 25 (0,4) (5,9) (6,3) (8,2) (10,13) (11,15) (12,7) (14,17) (16,1)
slot 26 (1,8) (2,5) (3,14) (4,12) (7,11) (9,0) (13,6) (15,10) (17,16)
slot 27 (0,17) (1,3) (6,9) (8,13) (10,4) (11,5) (12,2) (14,15) (16,7)
slot 28 (2,14) (3,15) (4,8) (5,0) (7,1) (9,16) (12,6) (13,11) (17,10)
slot 29 (0,12) (1,5) (3,4) (6,2) (10,8) (14,9) (15,7) (16,13) (17,11)
slot 30 (2,1) (4,7) (5,14) (8,16) (9,3) (10,0) (11,6) (12,17) (13,15)
slot 31 (0,14) (1,13) (3,8) (6,4) (7,5) (10,11) (15,2) (16,12) (17,9)
slot 32 (2,0) (4,15) (5,6) (8,7) (9,10) (11,16) (12,3) (13,17) (14,1)
slot 33 (0,13) (1,11) (3,2) (6,14) (7,9) (10,12) (15,8) (16,5) (17,4)

Πρόβλημα Early 5

Το αποτέλεσμα που βρήκαμε είναι (36, 3983) το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους:

Ισχυροί περιορισμοί
207 = 189 ικανοποιούνται + 18 παραβιάζονται
CA1: 1
CA4: 14
GA1: 4
BR1: 1
BR2: 16
Ασθενείς περιορισμοί
CA1: 18
CA2: 905
CA3: 1135
CA4: 945
SE1: 980

και ωρολόγιο πρόγραμμα:

slot 0 (1,12) (2,14) (6,11) (8,4) (9,10) (13,3) (15,7) (16,0) (17,5)
slot 1 (0,8) (3,6) (4,17) (5,2) (9,7) (10,1) (11,16) (12,13) (14,15)
slot 2 (2,11) (4,0) (6,1) (7,16) (8,12) (13,10) (14,5) (15,3) (17,9)
slot 3 (0,6) (1,7) (3,2) (9,8) (10,4) (11,14) (12,5) (13,17) (16,15)
slot 4 (2,8) (3,1) (4,13) (5,6) (7,0) (10,16) (12,9) (14,17) (15,11)
slot 5 (0,12) (1,4) (6,2) (7,10) (8,14) (9,11) (13,5) (16,3) (17,15)

slot 6 (4,3) (5,9) (6,15) (10,8) (11,0) (12,7) (13,1) (14,16) (17,2)
slot 7 (0,1) (2,12) (3,10) (7,17) (8,6) (9,14) (11,5) (15,13) (16,4)
slot 8 (1,2) (4,14) (5,3) (6,9) (8,7) (10,17) (12,16) (13,11) (15,0)
slot 9 (0,9) (1,15) (2,10) (5,4) (7,3) (11,8) (14,6) (16,13) (17,12)
slot 10 (0,5) (3,17) (4,12) (7,2) (8,15) (9,1) (10,11) (13,14) (16,6)
slot 11 (1,8) (2,0) (4,7) (5,16) (6,13) (11,17) (12,10) (14,3) (15,9)
slot 12 (0,3) (7,11) (9,4) (12,6) (13,2) (14,10) (15,5) (16,8) (17,1)
slot 13 (1,14) (2,15) (3,12) (4,11) (5,7) (6,17) (8,13) (9,16) (10,0)
slot 14 (2,4) (5,8) (6,7) (11,3) (13,9) (14,12) (15,10) (16,1) (17,0)
slot 15 (0,14) (1,11) (4,6) (7,13) (8,17) (9,3) (10,5) (12,15) (16,2)
slot 16 (2,9) (3,8) (5,1) (6,10) (11,12) (13,0) (14,7) (15,4) (17,16)
slot 17 (0,17) (1,3) (4,8) (7,14) (9,13) (11,10) (12,2) (15,6) (16,5)
slot 18 (2,1) (3,5) (6,12) (8,0) (9,15) (10,7) (13,16) (14,4) (17,11)
slot 19 (0,2) (1,10) (3,15) (4,9) (5,13) (7,8) (11,6) (12,17) (16,14)
slot 20 (2,3) (6,5) (7,12) (8,11) (9,0) (13,4) (14,1) (15,16) (17,10)
slot 21 (0,15) (1,13) (3,11) (5,14) (6,8) (9,17) (10,2) (12,4) (16,7)
slot 22 (0,16) (2,13) (4,5) (7,1) (10,3) (11,9) (14,8) (15,12) (17,6)
slot 23 (1,0) (3,9) (5,10) (6,4) (8,16) (11,2) (12,14) (13,15) (17,7)
slot 24 (0,13) (4,1) (7,6) (8,3) (9,5) (10,12) (14,11) (15,2) (16,17)
slot 25 (1,9) (2,7) (3,16) (4,10) (5,15) (6,0) (11,13) (12,8) (17,14)
slot 26 (0,10) (2,17) (3,4) (7,5) (8,1) (9,12) (13,6) (15,14) (16,11)
slot 27 (4,15) (5,0) (6,16) (8,9) (10,13) (11,7) (12,1) (14,2) (17,3)
slot 28 (0,4) (1,5) (2,16) (3,14) (7,9) (10,6) (12,11) (13,8) (15,17)
slot 29 (3,13) (5,12) (7,15) (8,2) (9,6) (11,1) (14,0) (16,10) (17,4)
slot 30 (0,11) (1,16) (4,2) (5,17) (6,14) (10,9) (12,3) (13,7) (15,8)
slot 31 (1,6) (2,5) (3,0) (7,4) (8,10) (11,15) (14,9) (16,12) (17,13)
slot 32 (0,7) (4,16) (5,11) (6,3) (9,2) (10,14) (13,12) (15,1) (17,8)
slot 33 (1,17) (2,6) (3,7) (8,5) (10,15) (11,4) (12,0) (14,13) (16,9)

Πρόβλημα Early 6

Βρήκαμε την λύση (0, 5630) που έχει ανάλυση κόστους

CA1: 15
CA2: 1675
CA3: 305
CA4: 815
BR2: 1500

FA2: 180

SE1: 1140

και ωρολόγιο πρόγραμμα:

slot 0 (1,15) (2,3) (6,5) (7,16) (8,17) (11,10) (12,0) (13,9) (14,4)
slot 1 (0,15) (4,7) (5,3) (8,6) (11,16) (12,9) (13,2) (14,10) (17,1)
slot 2 (0,17) (3,14) (4,12) (6,13) (7,11) (9,2) (10,1) (15,5) (16,8)
slot 3 (1,11) (2,10) (3,6) (5,14) (8,0) (9,4) (12,13) (15,7) (17,16)
slot 4 (0,3) (1,8) (5,10) (6,15) (7,12) (11,2) (13,4) (14,17) (16,9)
slot 5 (0,1) (2,17) (3,7) (4,11) (9,6) (10,12) (14,13) (15,8) (16,5)
slot 6 (2,0) (3,16) (7,14) (8,4) (9,15) (11,6) (12,1) (13,10) (17,5)
slot 7 (1,7) (5,4) (6,10) (8,9) (12,3) (14,0) (15,11) (16,2) (17,13)
slot 8 (0,6) (1,16) (2,7) (4,3) (5,8) (9,14) (10,17) (11,13) (15,12)
slot 9 (2,5) (3,10) (4,1) (6,12) (7,9) (13,8) (14,11) (16,0) (17,15)
slot 10 (0,4) (1,5) (3,13) (6,17) (8,14) (9,11) (10,7) (12,16) (15,2)
slot 11 (2,1) (4,15) (5,13) (7,0) (9,3) (11,8) (12,17) (14,6) (16,10)
slot 12 (2,12) (3,1) (5,0) (6,4) (8,7) (10,9) (13,15) (14,16) (17,11)
slot 13 (0,10) (1,6) (4,2) (7,17) (8,12) (9,5) (11,3) (15,14) (16,13)
slot 14 (3,15) (4,17) (5,7) (6,16) (9,1) (10,8) (12,11) (13,0) (14,2)
slot 15 (0,9) (1,13) (2,8) (7,6) (10,15) (11,5) (12,14) (16,4) (17,3)
slot 16 (0,11) (4,10) (5,12) (6,2) (7,13) (8,3) (14,1) (15,16) (17,9)
slot 17 (1,17) (2,14) (3,0) (4,8) (9,7) (12,6) (13,5) (15,10) (16,11)
slot 18 (1,9) (5,15) (6,8) (10,0) (11,4) (13,3) (14,7) (16,12) (17,2)
slot 19 (0,14) (3,17) (5,16) (6,11) (7,2) (8,10) (9,13) (12,4) (15,1)
slot 20 (0,5) (2,15) (3,8) (4,9) (11,1) (12,7) (13,14) (16,6) (17,10)
slot 21 (1,3) (2,16) (9,8) (10,6) (11,12) (13,7) (14,5) (15,0) (17,4)
slot 22 (0,13) (1,4) (3,12) (5,11) (6,9) (7,10) (8,2) (15,17) (16,14)
slot 23 (4,13) (6,1) (8,5) (10,3) (11,9) (12,2) (14,15) (16,7) (17,0)
slot 24 (0,7) (2,4) (3,11) (5,17) (9,12) (10,16) (13,1) (14,8) (15,6)
slot 25 (1,2) (4,6) (7,3) (8,13) (9,10) (11,0) (12,5) (16,15) (17,14)
slot 26 (1,12) (2,9) (4,5) (6,0) (8,15) (10,11) (13,16) (14,3) (17,7)
slot 27 (0,12) (2,13) (3,4) (5,6) (7,8) (10,14) (11,17) (15,9) (16,1)
slot 28 (1,14) (3,2) (4,16) (7,5) (9,0) (11,15) (12,10) (13,6) (17,8)
slot 29 (0,2) (5,1) (6,7) (8,11) (10,4) (13,12) (14,9) (15,3) (16,17)
slot 30 (0,16) (2,6) (3,9) (7,4) (8,1) (10,5) (11,14) (15,13) (17,12)
slot 31 (1,10) (4,0) (5,2) (6,14) (7,15) (9,17) (12,8) (13,11) (16,3)
slot 32 (1,0) (5,9) (6,3) (8,16) (10,2) (11,7) (13,17) (14,12) (15,4)
slot 33 (0,8) (2,11) (3,5) (4,14) (7,1) (9,16) (10,13) (12,15) (17,6)

Πρόβλημα Early 7

Βρήκαμε την λύση (0, 10818) που έχει ανάλυση κόστους

CA1: 30
CA2: 1815
CA3: 695
CA4: 2230
GA1: 18
SE1: 6030
GA1: 40

και ωρολόγιο πρόγραμμα:

slot 0 (0,9) (2,12) (3,17) (6,14) (7,15) (8,1) (10,4) (11,16) (13,5)
slot 1 (1,11) (3,13) (4,2) (5,7) (9,8) (12,0) (14,6) (15,17) (16,10)
slot 2 (0,15) (2,14) (6,3) (7,5) (8,16) (10,12) (11,1) (13,9) (17,4)
slot 3 (1,17) (2,8) (3,10) (6,7) (9,13) (12,4) (14,0) (15,11) (16,5)
slot 4 (0,6) (1,9) (4,16) (5,15) (8,3) (10,14) (11,13) (12,2) (17,7)
slot 5 (3,11) (5,4) (6,0) (7,2) (9,10) (13,17) (14,1) (15,12) (16,8)
slot 6 (0,14) (1,16) (2,3) (4,7) (8,13) (10,15) (11,9) (12,6) (17,5)
slot 7 (2,13) (3,1) (5,8) (6,17) (7,12) (9,0) (14,10) (15,4) (16,11)
slot 8 (0,16) (2,7) (4,5) (8,6) (10,3) (12,15) (13,1) (14,9) (17,11)
slot 9 (1,10) (3,14) (4,13) (5,0) (6,12) (7,17) (9,15) (11,8) (16,2)
slot 10 (0,7) (1,3) (2,6) (8,9) (10,13) (12,11) (14,4) (15,5) (17,16)
slot 11 (4,14) (5,10) (6,1) (7,8) (9,2) (11,3) (13,0) (16,12) (17,15)
slot 12 (0,2) (1,7) (3,6) (5,17) (10,8) (11,4) (12,9) (14,13) (15,16)
slot 13 (2,11) (4,1) (6,15) (7,0) (8,5) (9,12) (13,14) (16,3) (17,10)
slot 14 (0,17) (1,8) (3,16) (4,6) (5,13) (10,9) (11,7) (14,12) (15,2)
slot 15 (2,5) (3,15) (7,6) (8,11) (9,14) (12,10) (13,4) (16,1) (17,0)
slot 16 (0,10) (1,14) (4,17) (5,11) (6,2) (7,3) (8,15) (12,13) (16,9)
slot 17 (2,16) (3,12) (6,13) (9,4) (10,0) (11,5) (14,8) (15,7) (17,1)
slot 18 (0,3) (1,6) (4,9) (5,2) (7,10) (12,17) (13,11) (15,8) (16,14)
slot 19 (0,4) (2,1) (3,7) (8,12) (9,11) (10,5) (14,15) (16,6) (17,13)
slot 20 (1,2) (4,15) (5,3) (6,16) (7,9) (8,17) (11,0) (12,14) (13,10)
slot 21 (0,11) (3,2) (8,4) (9,5) (10,7) (13,6) (14,16) (15,1) (17,12)
slot 22 (1,0) (2,15) (5,14) (6,9) (7,4) (8,10) (11,17) (12,3) (16,13)
slot 23 (0,5) (3,8) (4,11) (9,1) (10,6) (12,7) (13,16) (15,14) (17,2)
slot 24 (1,4) (2,17) (5,9) (6,10) (7,13) (8,0) (11,12) (14,3) (16,15)
slot 25 (4,0) (7,11) (9,3) (10,16) (12,1) (13,2) (14,5) (15,6) (17,8)

slot 26 (0,12) (1,13) (2,10) (3,4) (6,5) (8,14) (9,17) (11,15) (16,7)
slot 27 (4,8) (5,6) (7,1) (10,2) (11,14) (12,16) (13,3) (15,0) (17,9)
slot 28 (1,15) (2,4) (3,5) (6,8) (9,7) (11,10) (13,12) (14,17) (16,0)
slot 29 (0,13) (4,12) (5,1) (7,14) (8,2) (9,16) (10,11) (15,3) (17,6)
slot 30 (1,5) (2,9) (3,0) (6,4) (12,8) (13,7) (14,11) (15,10) (16,17)
slot 31 (0,8) (4,3) (5,12) (7,16) (9,6) (10,1) (11,2) (13,15) (17,14)
slot 32 (0,1) (3,9) (6,11) (8,7) (10,17) (12,5) (14,2) (15,13) (16,4)
slot 33 (1,12) (2,0) (4,10) (5,16) (11,6) (13,8) (14,7) (15,9) (17,3)

Πρόβλημα Early 8

Βρήκαμε την λύση (0, 1994) που έχει ανάλυση κόστους

CA2: 105
CA3: 380
CA4: 1470
GA1: 39

και ωρολόγιο πρόγραμμα:

slot 0 (0,6) (4,11) (7,16) (8,5) (9,3) (12,2) (13,10) (14,1) (15,17)
slot 1 (1,4) (2,9) (3,14) (6,7) (8,15) (10,11) (12,0) (16,13) (17,5)
slot 2 (1,8) (4,9) (5,6) (7,10) (11,12) (13,2) (14,0) (16,15) (17,3)
slot 3 (0,15) (1,13) (3,2) (6,4) (9,17) (11,7) (12,5) (14,10) (16,8)
slot 4 (2,16) (5,3) (7,1) (8,6) (9,14) (10,4) (13,0) (15,12) (17,11)
slot 5 (1,17) (2,5) (3,16) (4,14) (8,7) (9,6) (10,15) (11,0) (13,12)
slot 6 (0,3) (1,2) (5,9) (6,10) (7,4) (12,8) (14,16) (15,11) (17,13)
slot 7 (0,16) (2,8) (4,10) (6,3) (7,5) (11,1) (12,17) (13,9) (15,14)
slot 8 (0,1) (2,6) (3,13) (4,15) (9,11) (10,12) (14,8) (16,5) (17,7)
slot 9 (3,1) (5,11) (6,15) (8,4) (9,0) (10,2) (13,7) (16,12) (17,14)
slot 10 (1,12) (4,16) (5,0) (6,14) (8,13) (10,7) (11,2) (15,3) (17,9)
slot 11 (0,8) (2,12) (3,6) (7,9) (10,1) (13,4) (14,11) (15,5) (16,17)
slot 12 (1,16) (4,0) (5,10) (6,13) (7,15) (11,3) (12,9) (14,2) (17,8)
slot 13 (0,17) (2,15) (4,12) (5,1) (8,3) (9,13) (10,16) (11,6) (14,7)
slot 14 (1,11) (2,17) (3,4) (5,13) (7,0) (8,14) (12,10) (15,9) (16,6)
slot 15 (0,10) (1,14) (2,7) (3,12) (4,5) (6,17) (9,16) (11,8) (13,15)
slot 16 (0,2) (1,6) (5,14) (9,8) (10,3) (12,7) (15,13) (16,11) (17,4)
slot 17 (0,7) (2,14) (3,5) (8,17) (9,10) (11,16) (12,4) (13,6) (15,1)
slot 18 (1,5) (3,15) (6,9) (7,2) (12,11) (13,8) (14,4) (16,10) (17,0)
slot 19 (4,2) (6,16) (7,11) (8,0) (9,5) (10,13) (12,15) (14,3) (17,1)

slot 20 (2,3) (4,1) (5,12) (7,6) (10,14) (11,9) (13,17) (15,8) (16,0)
slot 21 (2,4) (3,7) (5,16) (6,1) (8,12) (10,9) (11,17) (14,13) (15,0)
slot 22 (0,4) (2,13) (3,11) (5,8) (7,14) (9,12) (15,6) (16,1) (17,10)
slot 23 (1,0) (3,17) (4,7) (6,12) (9,2) (10,8) (11,15) (13,16) (14,5)
slot 24 (0,13) (4,3) (5,7) (8,1) (11,10) (12,14) (15,2) (16,9) (17,6)
slot 25 (0,12) (1,9) (5,4) (6,11) (8,10) (13,3) (14,17) (15,7) (16,2)
slot 26 (1,15) (7,3) (8,2) (10,0) (11,4) (12,6) (13,5) (14,9) (17,16)
slot 27 (0,14) (2,10) (3,8) (4,17) (6,5) (9,7) (11,13) (12,1) (15,16)
slot 28 (0,5) (1,3) (2,11) (6,8) (7,13) (9,4) (15,10) (16,14) (17,12)
slot 29 (1,10) (3,0) (4,6) (5,2) (8,9) (11,14) (12,13) (16,7) (17,15)
slot 30 (2,0) (4,8) (7,12) (9,15) (10,17) (11,5) (13,1) (14,6) (16,3)
slot 31 (0,9) (1,7) (3,10) (4,13) (5,17) (6,2) (8,11) (12,16) (14,15)
slot 32 (0,11) (5,15) (7,8) (9,1) (10,6) (12,3) (13,14) (16,4) (17,2)
slot 33 (2,1) (3,9) (6,0) (7,17) (8,16) (10,5) (13,11) (14,12) (15,4)

Πρόβλημα Early 9

Βρήκαμε την λύση (0, 1613) που έχει ανάλυση κόστους

CA3: 365
GA1: 3
BR1: 5
BR2: 1240

και ωρολόγιο πρόγραμμα:

slot 0 (1,15) (4,0) (5,14) (8,7) (9,17) (10,16) (11,2) (12,3) (13,6)
slot 1 (0,17) (1,4) (2,7) (3,10) (5,12) (6,9) (14,8) (15,11) (16,13)
slot 2 (6,0) (7,16) (8,2) (9,4) (10,1) (12,14) (13,11) (15,5) (17,3)
slot 3 (1,0) (2,6) (3,4) (5,9) (7,14) (10,12) (11,8) (16,15) (17,13)
slot 4 (0,7) (4,3) (6,2) (8,16) (9,10) (12,11) (13,5) (14,1) (15,17)
slot 5 (2,13) (3,9) (4,8) (5,11) (6,1) (7,0) (10,14) (12,15) (16,17)
slot 6 (0,10) (4,16) (8,1) (9,5) (11,12) (13,2) (14,6) (15,3) (17,7)
slot 7 (0,14) (3,11) (7,17) (8,15) (9,2) (10,5) (12,1) (13,4) (16,6)
slot 8 (0,9) (2,12) (3,1) (4,7) (5,13) (6,11) (14,10) (15,16) (17,8)
slot 9 (1,13) (5,6) (7,2) (8,0) (9,3) (10,17) (11,14) (15,4) (16,12)
slot 10 (0,5) (3,2) (6,10) (7,4) (11,15) (12,8) (13,9) (14,16) (17,1)
slot 11 (1,6) (2,15) (4,11) (5,8) (9,12) (10,0) (13,7) (14,17) (16,3)
slot 12 (0,16) (1,5) (2,11) (3,7) (4,14) (8,13) (12,10) (15,9) (17,6)
slot 13 (1,8) (3,16) (5,2) (7,13) (9,0) (11,6) (14,4) (15,10) (17,12)

slot 14 (0,15) (2,5) (4,1) (6,7) (10,8) (11,17) (12,9) (13,3) (16,14)
slot 15 (1,10) (3,6) (7,12) (8,4) (9,11) (13,16) (14,2) (15,0) (17,5)
slot 16 (2,17) (4,15) (5,0) (7,3) (10,6) (11,1) (12,13) (14,9) (16,8)
slot 17 (0,4) (1,7) (2,10) (3,8) (6,12) (9,16) (11,5) (13,14) (17,15)
slot 18 (4,13) (5,17) (6,3) (7,9) (8,12) (10,2) (14,11) (15,1) (16,0)
slot 19 (0,6) (1,9) (2,14) (5,3) (11,7) (12,4) (13,10) (15,8) (17,16)
slot 20 (0,8) (3,12) (4,5) (6,17) (7,10) (9,1) (11,13) (14,15) (16,2)
slot 21 (2,9) (3,15) (5,1) (8,6) (10,11) (13,12) (14,7) (16,4) (17,0)
slot 22 (0,3) (1,2) (4,9) (6,13) (8,17) (11,10) (12,5) (15,14) (16,7)
slot 23 (2,0) (5,4) (6,8) (7,1) (9,14) (11,3) (12,17) (13,15) (16,10)
slot 24 (0,13) (1,16) (4,6) (7,5) (8,9) (10,3) (14,12) (15,2) (17,11)
slot 25 (1,14) (2,4) (3,13) (5,15) (6,16) (9,8) (11,0) (12,7) (17,10)
slot 26 (0,12) (3,17) (4,2) (7,6) (8,11) (10,15) (13,1) (14,5) (16,9)
slot 27 (0,11) (1,3) (2,16) (5,10) (6,14) (9,7) (13,8) (15,12) (17,4)
slot 28 (3,0) (6,15) (8,5) (10,7) (11,4) (12,2) (14,13) (16,1) (17,9)
slot 29 (0,2) (1,11) (3,5) (4,10) (8,14) (9,6) (12,16) (13,17) (15,7)
slot 30 (1,17) (2,8) (5,7) (6,4) (10,9) (11,16) (12,0) (14,3) (15,13)
slot 31 (0,1) (3,14) (4,12) (6,5) (7,15) (8,10) (9,13) (16,11) (17,2)
slot 32 (1,12) (2,3) (5,16) (7,8) (10,4) (11,9) (13,0) (15,6) (17,14)
slot 33 (2,1) (4,17) (7,11) (8,3) (9,15) (10,13) (12,6) (14,0) (16,5)

Πρόβλημα Early 10

Το αποτέλεσμα που βρήκαμε είναι (34, 4899) το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους:

Ισχυροί περιορισμοί
246 = 240 ικανοποιούνται + 6 παραβιάζονται
CA3: 3
CA4: 5
BR2: 26
Ασθενείς περιορισμοί
CA1: 24
CA2: 1525
CA3: 165
CA4: 2075
SE1: 1110

και ωρολόγιο πρόγραμμα:

slot 0 (0,6) (5,13) (8,18) (9,4) (10,1) (11,19) (12,15) (14,2) (16,3) (17,7)
slot 1 (1,9) (2,12) (4,3) (5,19) (6,14) (7,10) (13,0) (15,8) (17,16) (18,11)
slot 2 (0,5) (2,13) (3,7) (4,18) (9,6) (10,16) (11,15) (12,8) (14,17) (19,1)
slot 3 (1,6) (4,19) (8,7) (9,5) (11,3) (13,10) (14,12) (16,15) (17,2) (18,0)
slot 4 (0,8) (1,17) (5,2) (6,10) (7,14) (12,3) (13,11) (15,4) (16,9) (19,18)
slot 5 (0,9) (2,16) (3,6) (4,11) (7,18) (8,5) (10,12) (14,1) (17,13) (19,15)
slot 6 (0,14) (1,2) (6,7) (9,3) (11,5) (12,4) (13,19) (15,17) (16,8) (18,10)
slot 7 (2,0) (3,18) (4,1) (5,12) (7,11) (8,13) (10,15) (14,9) (17,6) (19,16)
slot 8 (0,10) (1,3) (2,7) (6,8) (9,17) (11,16) (12,19) (13,4) (14,18) (15,5)
slot 9 (3,0) (4,2) (5,6) (7,1) (8,11) (10,17) (15,9) (16,13) (18,12) (19,14)
slot 10 (0,16) (1,15) (2,19) (3,10) (5,7) (6,4) (9,18) (12,11) (14,13) (17,8)
slot 11 (1,16) (7,0) (8,4) (10,2) (11,9) (13,6) (15,14) (17,12) (18,5) (19,3)
slot 12 (0,11) (2,8) (3,13) (4,17) (5,1) (6,15) (7,19) (12,9) (14,10) (16,18)
slot 13 (1,0) (3,2) (5,4) (8,10) (9,7) (11,14) (13,12) (15,18) (16,6) (19,17)
slot 14 (2,9) (4,7) (10,5) (11,1) (12,6) (14,16) (15,3) (17,0) (18,13) (19,8)
slot 15 (0,19) (3,17) (5,14) (6,11) (7,12) (8,1) (9,10) (13,15) (16,4) (18,2)
slot 16 (1,18) (2,15) (5,3) (7,16) (9,13) (10,4) (12,0) (14,8) (17,11) (19,6)
slot 17 (3,14) (4,0) (6,18) (8,9) (11,2) (13,1) (15,7) (16,12) (17,5) (19,10)
slot 18 (0,15) (1,12) (2,6) (4,14) (5,16) (7,13) (8,3) (9,19) (10,11) (18,17)
slot 19 (3,1) (6,5) (8,0) (10,9) (12,2) (13,16) (14,7) (15,11) (17,19) (18,4)
slot 20 (0,3) (1,14) (2,18) (4,6) (5,10) (9,8) (11,12) (15,13) (16,17) (19,7)
slot 21 (3,4) (6,19) (7,5) (8,2) (10,0) (11,17) (12,1) (13,9) (14,15) (18,16)
slot 22 (2,3) (4,12) (5,8) (6,13) (9,14) (10,18) (15,0) (16,7) (17,1) (19,11)
slot 23 (0,17) (1,10) (3,16) (4,9) (7,15) (11,6) (12,5) (13,2) (14,19) (18,8)
slot 24 (2,5) (6,3) (8,17) (9,12) (10,13) (11,7) (15,1) (16,0) (18,14) (19,4)
slot 25 (0,4) (1,19) (3,8) (5,11) (7,6) (12,14) (13,18) (15,2) (16,10) (17,9)
slot 26 (2,10) (4,15) (6,12) (7,17) (8,16) (9,1) (11,13) (14,5) (18,3) (19,0)
slot 27 (0,2) (1,11) (3,9) (5,15) (8,6) (10,14) (12,18) (13,7) (16,19) (17,4)
slot 28 (2,14) (4,5) (6,16) (7,8) (11,0) (13,3) (15,12) (17,10) (18,1) (19,9)
slot 29 (0,7) (1,13) (3,11) (5,18) (8,19) (9,15) (10,6) (12,17) (14,4) (16,2)
slot 30 (2,1) (6,0) (7,4) (11,8) (12,16) (13,14) (15,10) (17,3) (18,9) (19,5)
slot 31 (0,12) (1,5) (3,19) (4,8) (9,2) (10,7) (13,17) (14,6) (16,11) (18,15)
slot 32 (0,13) (5,9) (6,1) (7,3) (8,15) (11,4) (12,10) (16,14) (17,18) (19,2)
slot 33 (1,7) (2,17) (3,12) (4,16) (9,11) (10,8) (13,5) (14,0) (15,6) (18,19)
slot 34 (0,1) (3,15) (4,10) (6,9) (7,2) (8,12) (11,18) (16,5) (17,14) (19,13)
slot 35 (1,4) (2,11) (5,17) (9,0) (10,19) (12,7) (13,8) (14,3) (15,16) (18,6)
slot 36 (0,18) (3,5) (4,13) (6,2) (7,9) (8,14) (11,10) (16,1) (17,15) (19,12)
slot 37 (1,8) (2,4) (5,0) (6,17) (9,16) (10,3) (12,13) (14,11) (15,19) (18,7)

Πρόβλημα Early 11

Βρήκαμε την λύση (0, 11442) που έχει ανάλυση κόστους

CA1: 26
CA2: 1545
CA3: 640
CA4: 2120
GA1: 1
SE1: 7110

και ωρολόγιο πρόγραμμα:

slot 0 (1,18) (4,2) (7,3) (8,0) (9,5) (10,11) (12,6) (13,19) (14,16) (15,17)
slot 1 (0,15) (1,13) (3,9) (5,2) (6,8) (11,10) (16,14) (17,4) (18,7) (19,12)
slot 2 (2,0) (4,5) (7,16) (8,10) (9,1) (12,11) (13,6) (14,17) (15,18) (19,3)
slot 3 (0,12) (1,19) (5,9) (6,7) (10,4) (11,8) (14,13) (16,15) (17,3) (18,2)
slot 4 (2,5) (3,1) (4,16) (7,11) (9,14) (12,8) (13,10) (15,6) (18,17) (19,0)
slot 5 (1,4) (5,18) (6,12) (8,19) (10,13) (11,7) (14,2) (15,3) (16,0) (17,9)
slot 6 (0,11) (3,17) (4,6) (7,15) (8,1) (10,19) (12,2) (13,16) (14,9) (18,5)
slot 7 (1,14) (2,12) (5,0) (6,10) (9,15) (13,3) (16,11) (17,7) (18,4) (19,8)
slot 8 (0,14) (3,18) (4,19) (7,13) (8,16) (10,6) (11,2) (12,1) (15,9) (17,5)
slot 9 (1,0) (2,6) (3,11) (5,4) (7,17) (9,12) (14,10) (16,8) (18,13) (19,15)
slot 10 (0,8) (1,3) (2,9) (4,14) (6,19) (10,18) (11,5) (12,7) (13,17) (15,16)
slot 11 (3,2) (5,1) (7,0) (8,11) (9,6) (12,10) (14,4) (17,13) (18,15) (19,16)
slot 12 (0,5) (1,12) (2,15) (4,8) (6,3) (10,9) (11,17) (13,7) (14,18) (16,19)
slot 13 (0,6) (3,13) (5,16) (7,4) (8,12) (9,11) (15,14) (17,2) (18,10) (19,1)
slot 14 (2,1) (4,3) (6,18) (7,19) (10,8) (11,14) (12,15) (13,5) (16,9) (17,0)
slot 15 (0,13) (1,2) (5,6) (8,17) (9,10) (14,11) (15,12) (16,7) (18,3) (19,4)
slot 16 (2,14) (3,0) (5,19) (6,16) (9,7) (10,12) (11,4) (13,18) (15,1) (17,8)
slot 17 (1,9) (4,17) (6,13) (7,18) (8,2) (11,15) (12,0) (14,3) (16,5) (19,10)
slot 18 (0,7) (1,6) (2,17) (3,16) (5,8) (9,4) (10,14) (15,13) (18,12) (19,11)
slot 19 (6,2) (7,9) (8,5) (10,15) (11,19) (12,4) (13,0) (14,1) (16,3) (17,18)
slot 20 (0,10) (1,17) (2,13) (3,14) (4,18) (5,7) (9,8) (15,11) (16,12) (19,6)
slot 21 (0,16) (6,4) (7,2) (8,14) (10,5) (11,3) (12,9) (13,15) (17,1) (18,19)
slot 22 (1,10) (2,8) (3,19) (4,13) (7,6) (9,0) (11,18) (14,12) (15,5) (16,17)
slot 23 (0,9) (2,7) (5,15) (6,11) (8,4) (10,16) (12,3) (13,1) (17,14) (19,18)
slot 24 (1,5) (3,10) (4,12) (9,13) (11,0) (14,6) (15,7) (16,2) (17,19) (18,8)
slot 25 (0,17) (6,15) (7,1) (8,9) (10,3) (12,5) (13,2) (16,4) (18,11) (19,14)
slot 26 (1,15) (2,19) (3,6) (4,7) (5,10) (8,18) (9,17) (11,13) (12,16) (14,0)

slot 27 (0,4) (2,3) (6,9) (7,8) (13,12) (14,5) (15,19) (16,10) (17,11) (18,1)
slot 28 (0,2) (3,15) (4,1) (5,14) (8,7) (9,16) (10,17) (11,6) (12,18) (19,13)
slot 29 (1,8) (2,11) (3,4) (6,5) (7,12) (13,9) (14,19) (15,10) (17,16) (18,0)
slot 30 (0,3) (4,10) (5,13) (6,14) (8,15) (9,2) (11,1) (12,17) (16,18) (19,7)
slot 31 (1,7) (2,16) (3,5) (10,0) (13,11) (14,8) (15,4) (17,12) (18,6) (19,9)
slot 32 (0,1) (2,18) (4,11) (5,3) (7,10) (8,13) (9,19) (12,14) (16,6) (17,15)
slot 33 (1,16) (3,8) (6,17) (10,2) (11,12) (13,4) (14,7) (15,0) (18,9) (19,5)
slot 34 (0,19) (2,10) (4,15) (5,11) (7,14) (8,3) (9,18) (12,13) (16,1) (17,6)
slot 35 (3,7) (4,9) (5,12) (6,0) (10,1) (11,16) (13,8) (15,2) (18,14) (19,17)
slot 36 (0,18) (1,11) (2,4) (7,5) (8,6) (9,3) (12,19) (14,15) (16,13) (17,10)
slot 37 (3,12) (4,0) (5,17) (6,1) (10,7) (11,9) (13,14) (15,8) (18,16) (19,2)

Πρόβλημα Early 12

Βρήκαμε την λύση (0, 1445) που έχει ανάλυση κόστους

CA3: 35
BR1: 10
BR2: 1400

και ωρολόγιο πρόγραμμα:

slot 0 (0,16) (2,14) (8,4) (11,10) (12,3) (13,9) (15,1) (17,5) (18,7) (19,6)
slot 1 (0,15) (1,13) (3,11) (5,2) (6,8) (7,19) (9,10) (12,17) (14,18) (16,4)
slot 2 (2,3) (4,19) (7,1) (8,13) (9,12) (10,17) (11,5) (15,6) (16,14) (18,0)
slot 3 (0,8) (2,16) (5,9) (6,18) (11,15) (12,10) (13,4) (14,3) (17,7) (19,1)
slot 4 (0,2) (1,14) (3,10) (4,11) (6,9) (7,13) (8,16) (12,5) (15,19) (18,17)
slot 5 (1,6) (2,8) (4,12) (9,11) (10,7) (13,15) (14,5) (16,18) (17,0) (19,3)
slot 6 (0,4) (2,15) (5,1) (8,3) (10,6) (11,7) (14,9) (16,12) (17,13) (18,19)
slot 7 (1,18) (3,0) (4,10) (6,2) (7,5) (9,16) (11,17) (12,8) (15,14) (19,13)
slot 8 (0,10) (2,19) (5,4) (6,12) (8,1) (13,11) (14,17) (15,9) (16,7) (18,3)
slot 9 (3,1) (4,14) (5,15) (7,6) (8,19) (9,18) (10,16) (11,0) (12,13) (17,2)
slot 10 (0,14) (1,4) (3,16) (6,5) (9,7) (12,2) (13,10) (15,17) (18,8) (19,11)
slot 11 (1,10) (2,9) (4,6) (5,3) (7,12) (8,15) (14,11) (16,17) (18,13) (19,0)
slot 12 (0,5) (3,7) (6,16) (9,8) (10,2) (11,18) (12,19) (13,14) (15,4) (17,1)
slot 13 (1,12) (2,4) (5,10) (7,0) (13,3) (14,8) (16,11) (17,6) (18,15) (19,9)
slot 14 (3,15) (4,17) (6,0) (7,14) (8,5) (9,1) (10,19) (11,2) (12,18) (16,13)
slot 15 (0,12) (1,11) (4,7) (5,16) (9,3) (13,6) (14,19) (15,10) (17,8) (18,2)
slot 16 (0,9) (2,13) (3,17) (6,11) (8,7) (10,14) (12,15) (16,1) (18,4) (19,5)
slot 17 (1,2) (4,3) (5,18) (7,15) (10,8) (11,12) (13,0) (14,6) (16,19) (17,9)

slot 18 (0,1) (2,7) (3,6) (5,13) (8,11) (9,4) (14,12) (15,16) (18,10) (19,17)
slot 19 (1,7) (3,5) (6,14) (10,4) (11,8) (12,0) (13,19) (16,2) (17,15) (18,9)
slot 20 (4,1) (5,17) (7,3) (8,6) (9,13) (10,18) (12,11) (14,2) (16,0) (19,15)
slot 21 (0,18) (1,9) (2,5) (6,4) (7,10) (11,3) (13,16) (15,12) (17,14) (19,8)
slot 22 (1,0) (3,13) (4,2) (5,19) (8,17) (10,9) (12,16) (14,7) (15,11) (18,6)
slot 23 (0,17) (2,10) (6,13) (7,8) (9,5) (11,1) (14,15) (16,3) (18,12) (19,4)
slot 24 (1,16) (3,2) (4,18) (6,19) (8,9) (10,0) (12,14) (13,7) (15,5) (17,11)
slot 25 (0,11) (2,1) (5,6) (7,4) (10,3) (14,13) (15,8) (16,9) (17,12) (19,18)
slot 26 (1,15) (3,19) (4,0) (6,17) (8,10) (9,2) (11,14) (12,7) (13,5) (18,16)
slot 27 (0,19) (1,8) (3,12) (7,9) (10,5) (11,13) (14,4) (15,2) (16,6) (17,18)
slot 28 (2,17) (4,15) (5,14) (6,3) (7,16) (8,12) (9,0) (13,1) (18,11) (19,10)
slot 29 (3,8) (4,13) (5,0) (6,7) (11,16) (12,9) (14,1) (15,18) (17,10) (19,2)
slot 30 (0,3) (1,19) (2,6) (7,17) (9,15) (10,11) (12,4) (13,8) (16,5) (18,14)
slot 31 (2,0) (3,18) (4,9) (5,12) (8,14) (10,1) (11,6) (15,13) (17,16) (19,7)
slot 32 (5,8) (6,10) (7,18) (9,19) (11,4) (12,1) (13,2) (14,0) (16,15) (17,3)
slot 33 (0,13) (1,17) (2,11) (3,14) (4,8) (9,6) (15,7) (16,10) (18,5) (19,12)
slot 34 (2,18) (3,9) (5,7) (6,1) (8,0) (10,15) (11,19) (13,12) (14,16) (17,4)
slot 35 (1,3) (4,5) (7,11) (8,2) (9,17) (12,6) (13,18) (14,10) (15,0) (19,16)
slot 36 (0,7) (2,12) (3,4) (5,11) (6,15) (9,14) (10,13) (16,8) (17,19) (18,1)
slot 37 (0,6) (1,5) (4,16) (7,2) (8,18) (10,12) (11,9) (13,17) (15,3) (19,14)

Πρόβλημα Early 13

Βρήκαμε την λύση (0, 730) που έχει ανάλυση κόστους

CA1: 19
CA2: 455
CA3: 240
GA1: 11
BR1: 5

και ωρολόγιο πρόγραμμα:

slot 0 (1,3) (2,7) (5,0) (9,14) (10,8) (11,16) (12,18) (13,17) (15,4) (19,6)
slot 1 (0,13) (3,10) (4,12) (6,19) (7,5) (8,2) (9,11) (14,1) (16,15) (17,18)
slot 2 (1,4) (2,19) (3,17) (5,7) (8,9) (10,0) (11,6) (12,15) (16,13) (18,14)
slot 3 (4,11) (6,0) (7,3) (9,5) (10,18) (13,2) (14,8) (15,16) (17,1) (19,12)
slot 4 (0,7) (1,15) (2,10) (5,17) (8,13) (12,11) (14,9) (16,6) (18,4) (19,3)
slot 5 (0,8) (3,16) (4,18) (6,14) (7,2) (10,5) (11,1) (13,9) (15,12) (17,19)

slot 6 (2,3) (6,1) (8,10) (9,13) (11,5) (12,17) (14,15) (16,4) (18,7) (19,0)
slot 7 (0,18) (1,11) (3,9) (5,12) (7,19) (10,4) (13,8) (14,16) (15,2) (17,6)
slot 8 (2,17) (4,3) (6,13) (7,0) (9,8) (11,15) (12,1) (16,5) (18,10) (19,14)
slot 9 (0,6) (2,9) (3,18) (5,19) (7,11) (8,4) (10,1) (13,15) (14,12) (17,16)
slot 10 (1,0) (4,2) (8,3) (9,6) (11,17) (12,7) (13,14) (15,18) (16,10) (19,5)
slot 11 (0,16) (2,1) (5,11) (6,12) (7,4) (10,3) (14,13) (15,17) (18,9) (19,8)
slot 12 (1,7) (3,0) (4,5) (8,15) (9,2) (11,18) (12,19) (13,6) (16,14) (17,10)
slot 13 (0,10) (5,3) (6,11) (7,1) (12,8) (14,2) (15,9) (17,4) (18,16) (19,13)
slot 14 (1,12) (2,15) (3,6) (4,14) (8,5) (9,17) (10,7) (11,19) (13,0) (16,18)
slot 15 (0,11) (4,6) (5,16) (7,13) (12,9) (14,3) (15,1) (17,8) (18,2) (19,10)
slot 16 (0,15) (1,17) (2,4) (3,7) (6,5) (8,19) (9,18) (10,16) (11,14) (13,12)
slot 17 (3,8) (4,1) (5,13) (6,7) (12,14) (15,10) (16,2) (17,11) (18,0) (19,9)
slot 18 (1,16) (2,12) (3,11) (4,15) (7,18) (8,6) (9,0) (10,17) (13,19) (14,5)
slot 19 (0,9) (5,10) (6,2) (11,7) (12,4) (15,13) (16,3) (17,14) (18,8) (19,1)
slot 20 (1,6) (2,16) (3,5) (7,12) (8,17) (9,4) (10,15) (11,0) (13,18) (14,19)
slot 21 (0,3) (4,8) (5,2) (6,10) (7,9) (15,14) (16,1) (17,13) (18,12) (19,11)
slot 22 (1,9) (2,0) (4,19) (6,18) (8,16) (11,3) (12,5) (13,7) (14,10) (17,15)
slot 23 (0,1) (3,13) (5,4) (7,6) (9,15) (10,14) (11,8) (16,12) (18,17) (19,2)
slot 24 (2,6) (4,16) (8,11) (9,7) (12,10) (13,1) (14,0) (15,3) (17,5) (19,18)
slot 25 (0,4) (1,2) (3,14) (5,8) (6,17) (7,15) (10,12) (11,9) (16,19) (18,13)
slot 26 (2,11) (4,17) (8,7) (9,1) (12,0) (13,10) (14,6) (15,5) (18,3) (19,16)
slot 27 (0,14) (1,19) (3,15) (4,13) (5,18) (6,9) (10,2) (11,12) (16,8) (17,7)
slot 28 (2,14) (7,16) (8,1) (9,10) (12,6) (13,11) (15,0) (17,3) (18,5) (19,4)
slot 29 (0,12) (1,18) (3,4) (5,9) (6,8) (10,13) (11,2) (14,7) (16,17) (19,15)
slot 30 (0,17) (1,5) (4,10) (6,16) (7,14) (9,19) (12,2) (13,3) (15,8) (18,11)
slot 31 (2,5) (3,1) (6,4) (8,12) (10,19) (11,13) (14,18) (15,7) (16,0) (17,9)
slot 32 (0,2) (1,10) (5,15) (7,8) (9,16) (12,3) (13,4) (14,11) (18,6) (19,17)
slot 33 (1,14) (2,13) (3,19) (4,0) (5,6) (8,18) (10,9) (15,11) (16,7) (17,12)
slot 34 (0,19) (2,8) (6,15) (7,10) (9,3) (11,4) (12,16) (13,5) (14,17) (18,1)
slot 35 (1,13) (3,12) (4,9) (5,14) (8,0) (10,6) (16,11) (17,2) (18,15) (19,7)
slot 36 (0,5) (1,8) (2,18) (6,3) (7,17) (9,12) (11,10) (13,16) (14,4) (15,19)
slot 37 (3,2) (4,7) (5,1) (8,14) (10,11) (12,13) (15,6) (16,9) (17,0) (18,19)

Πρόβλημα Early 14

Βρήκαμε την λύση (0, 1392) που έχει ανάλυση κόστους

CA1: 12

BR2: 1380

και ωρολόγιο πρόγραμμα:

slot 0 (1,7) (2,0) (4,6) (5,18) (13,9) (14,8) (15,12) (16,10) (17,11) (19,3)
slot 1 (0,2) (3,13) (5,15) (6,18) (7,19) (8,12) (9,14) (10,1) (11,4) (16,17)
slot 2 (1,0) (2,8) (6,10) (7,5) (12,11) (13,4) (14,9) (15,3) (18,17) (19,16)
slot 3 (0,15) (3,1) (4,12) (5,2) (9,6) (10,18) (11,8) (16,7) (17,13) (19,14)
slot 4 (1,16) (2,11) (6,3) (7,4) (8,14) (9,19) (10,15) (12,5) (13,17) (18,0)
slot 5 (0,19) (2,9) (3,12) (4,7) (11,5) (14,1) (15,10) (16,8) (17,6) (18,13)
slot 6 (1,18) (3,16) (5,19) (6,0) (7,2) (9,11) (12,10) (13,8) (14,4) (15,17)
slot 7 (1,12) (2,16) (4,3) (5,14) (8,13) (9,0) (10,7) (15,11) (17,19) (18,6)
slot 8 (0,5) (3,18) (4,9) (6,14) (7,1) (8,2) (11,13) (16,12) (17,10) (19,15)
slot 9 (1,5) (2,7) (3,4) (8,11) (10,6) (12,17) (13,0) (14,16) (15,19) (18,9)
slot 10 (0,10) (4,8) (5,1) (9,15) (11,18) (12,7) (14,6) (16,3) (17,2) (19,13)
slot 11 (0,4) (2,17) (6,16) (7,14) (8,1) (10,9) (11,12) (13,3) (15,5) (19,18)
slot 12 (0,7) (3,19) (4,2) (5,6) (8,15) (9,10) (12,13) (14,17) (16,1) (18,11)
slot 13 (0,8) (1,15) (2,18) (3,7) (6,13) (10,4) (11,16) (14,12) (17,5) (19,9)
slot 14 (4,11) (5,3) (7,0) (8,10) (9,16) (12,6) (13,1) (15,14) (18,2) (19,17)
slot 15 (1,2) (3,8) (6,12) (9,4) (10,13) (11,19) (14,7) (16,5) (17,0) (18,15)
slot 16 (0,9) (1,6) (2,3) (4,15) (5,11) (7,17) (8,19) (12,18) (13,10) (16,14)
slot 17 (2,10) (8,0) (9,3) (12,4) (13,6) (14,11) (15,7) (17,16) (18,1) (19,5)
slot 18 (0,18) (1,17) (3,9) (5,4) (6,19) (7,13) (10,8) (11,15) (12,14) (16,2)
slot 19 (4,16) (8,5) (9,1) (10,12) (13,11) (14,2) (15,0) (17,3) (18,7) (19,6)
slot 20 (1,13) (2,14) (3,17) (4,10) (5,12) (6,15) (7,8) (11,0) (16,9) (18,19)
slot 21 (0,13) (1,11) (6,5) (8,18) (9,2) (12,16) (14,3) (15,4) (17,7) (19,10)
slot 22 (2,19) (3,14) (4,1) (5,8) (7,18) (10,17) (11,6) (13,12) (15,9) (16,0)
slot 23 (0,3) (1,14) (5,7) (6,2) (8,4) (9,12) (13,16) (17,15) (18,10) (19,11)
slot 24 (2,6) (3,0) (4,5) (7,9) (10,19) (11,17) (12,1) (14,18) (15,8) (16,13)
slot 25 (5,0) (6,1) (7,10) (8,9) (11,2) (12,3) (13,15) (14,19) (17,4) (18,16)
slot 26 (0,12) (1,3) (2,5) (4,18) (6,8) (9,13) (10,11) (16,15) (17,14) (19,7)
slot 27 (1,8) (3,2) (5,17) (11,10) (12,9) (13,7) (14,0) (15,6) (16,19) (18,4)
slot 28 (0,17) (4,14) (6,11) (7,3) (9,8) (10,5) (13,2) (15,16) (18,12) (19,1)
slot 29 (0,11) (1,4) (2,13) (3,15) (5,9) (8,7) (12,19) (14,10) (16,6) (17,18)
slot 30 (5,16) (6,4) (7,12) (9,17) (10,0) (11,1) (13,14) (15,2) (18,3) (19,8)

slot 31 (1,9) (2,15) (3,5) (4,0) (6,17) (10,16) (11,7) (12,8) (13,19) (18,14)
slot 32 (0,6) (3,10) (7,11) (9,5) (12,2) (14,13) (15,1) (16,18) (17,8) (19,4)
slot 33 (0,16) (1,19) (2,12) (4,17) (5,10) (6,9) (7,15) (8,3) (11,14) (13,18)
slot 34 (3,6) (4,13) (9,7) (10,2) (12,15) (14,5) (16,11) (17,1) (18,8) (19,0)
slot 35 (0,1) (2,4) (7,6) (8,16) (10,14) (11,3) (13,5) (15,18) (17,9) (19,12)
slot 36 (1,10) (3,11) (5,13) (6,7) (8,17) (9,18) (12,0) (14,15) (16,4) (19,2)
slot 37 (0,14) (2,1) (4,19) (7,16) (8,6) (10,3) (11,9) (15,13) (17,12) (18,5)

Πρόβλημα Early 15

Βρήκαμε την λύση (0, 6681) που έχει ανάλυση κόστους

CA2: 1605
CA3: 580
CA4: 2285
GA1: 51
BR1: 20
BR2: 1820
FA2: 320

και ωρολόγιο πρόγραμμα:

slot 0 (0,10) (4,5) (6,14) (7,16) (11,2) (12,9) (13,3) (15,1) (17,19) (18,8)
slot 1 (1,4) (2,0) (3,7) (5,18) (6,15) (9,13) (12,11) (14,17) (16,10) (19,8)
slot 2 (1,6) (4,14) (5,3) (8,16) (10,11) (13,0) (15,2) (17,7) (18,12) (19,9)
slot 3 (0,1) (3,18) (6,2) (7,9) (8,14) (10,4) (11,13) (12,17) (15,19) (16,5)
slot 4 (1,11) (2,17) (4,8) (6,10) (7,5) (9,3) (12,13) (14,19) (16,0) (18,15)
slot 5 (0,12) (2,14) (3,10) (5,6) (9,4) (11,7) (13,8) (15,16) (17,18) (19,1)
slot 6 (0,19) (5,2) (6,7) (8,12) (10,1) (11,9) (14,18) (15,13) (16,3) (17,4)
slot 7 (1,16) (3,6) (4,15) (7,14) (8,11) (9,2) (10,19) (12,5) (13,17) (18,0)
slot 8 (0,14) (1,8) (2,18) (6,4) (7,13) (9,16) (11,5) (15,12) (17,10) (19,3)
slot 9 (3,1) (4,19) (5,0) (8,2) (10,15) (11,18) (12,7) (13,6) (14,9) (16,17)
slot 10 (0,4) (1,12) (2,10) (3,14) (6,9) (8,15) (16,11) (17,5) (18,7) (19,13)
slot 11 (1,2) (4,11) (5,19) (6,16) (7,8) (9,18) (12,14) (13,10) (15,3) (17,0)
slot 12 (3,17) (4,13) (5,9) (8,0) (10,7) (11,6) (12,16) (14,15) (18,1) (19,2)
slot 13 (0,11) (2,4) (3,12) (6,8) (7,19) (9,15) (10,5) (13,14) (16,18) (17,1)
slot 14 (1,14) (4,18) (5,13) (7,2) (8,10) (11,3) (12,6) (15,0) (17,9) (19,16)
slot 15 (0,7) (2,3) (4,12) (5,15) (6,19) (9,8) (11,17) (13,1) (14,16) (18,10)
slot 16 (1,5) (2,13) (3,0) (8,17) (10,9) (14,11) (15,7) (16,4) (18,6) (19,12)
slot 17 (0,6) (1,9) (3,8) (5,14) (7,4) (12,10) (13,18) (16,2) (17,15) (19,11)

slot 18 (4,3) (6,17) (7,1) (8,5) (9,0) (11,15) (12,2) (13,16) (14,10) (18,19)
slot 19 (0,13) (1,18) (2,19) (4,7) (5,12) (9,17) (10,3) (11,8) (15,6) (16,14)
slot 20 (3,9) (5,10) (7,0) (8,4) (13,12) (14,1) (15,11) (17,16) (18,2) (19,6)
slot 21 (0,16) (2,8) (4,9) (6,3) (11,1) (12,15) (13,5) (14,7) (18,17) (19,10)
slot 22 (1,13) (2,6) (3,5) (7,18) (9,14) (10,12) (11,0) (15,4) (16,19) (17,8)
slot 23 (4,16) (5,11) (8,9) (10,6) (12,1) (13,7) (14,0) (17,2) (18,3) (19,15)
slot 24 (0,17) (2,1) (3,13) (6,12) (7,10) (9,19) (11,4) (14,5) (15,18) (16,8)
slot 25 (1,7) (4,2) (6,0) (10,14) (12,8) (13,19) (15,5) (16,9) (17,3) (18,11)
slot 26 (0,2) (3,4) (5,17) (7,12) (8,6) (9,1) (10,18) (11,16) (13,15) (19,14)
slot 27 (0,3) (2,5) (4,6) (8,7) (12,19) (14,13) (15,10) (16,1) (17,11) (18,9)
slot 28 (1,15) (2,12) (3,16) (6,11) (7,17) (9,5) (10,8) (13,4) (18,14) (19,0)
slot 29 (0,8) (1,3) (4,17) (5,7) (9,6) (10,13) (11,12) (14,2) (16,15) (19,18)
slot 30 (2,11) (3,19) (4,10) (5,1) (8,18) (12,0) (14,6) (15,9) (16,7) (17,13)
slot 31 (1,0) (6,5) (7,3) (8,19) (9,12) (10,16) (11,14) (13,2) (15,17) (18,4)
slot 32 (0,15) (1,10) (2,9) (3,11) (5,4) (14,8) (16,6) (17,12) (18,13) (19,7)
slot 33 (0,9) (4,1) (5,16) (7,6) (10,2) (12,18) (13,11) (14,3) (15,8) (19,17)
slot 34 (2,15) (6,1) (8,3) (9,7) (10,0) (11,19) (12,4) (16,13) (17,14) (18,5)
slot 35 (0,18) (1,17) (3,2) (5,8) (6,13) (7,11) (9,10) (15,14) (16,12) (19,4)
slot 36 (2,7) (3,15) (4,0) (8,1) (11,10) (13,9) (14,12) (17,6) (18,16) (19,5)
slot 37 (0,5) (1,19) (2,16) (6,18) (7,15) (8,13) (9,11) (10,17) (12,3) (14,4)

Πρόβλημα Middle 1

Το αποτέλεσμα που βρήκαμε είναι (12, 6185) το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους:

Ισχυροί περιορισμοί
144 = 133 ικανοποιούνται + 11 παραβιάζονται
CA4: 11
BR1: 1
Ασθενείς περιορισμοί
CA1: 20
CA2: 2350
CA4: 2815
SE1: 1000

και ωρολόγιο πρόγραμμα:

slot 0 (0,1) (3,10) (5,8) (6,4) (9,2) (11,15) (13,12) (14,7)

slot 1 (1,6) (4,11) (5,14) (7,3) (8,9) (10,0) (12,2) (15,13)
slot 2 (0,15) (2,4) (6,8) (7,12) (9,3) (11,5) (13,10) (14,1)
slot 3 (1,2) (3,12) (4,9) (5,13) (6,7) (8,14) (10,15) (11,0)
slot 4 (0,13) (1,5) (3,14) (6,10) (7,4) (9,11) (12,8) (15,2)
slot 5 (1,12) (2,6) (4,0) (5,3) (8,7) (9,10) (11,13) (14,15)
slot 6 (0,8) (3,6) (7,5) (10,4) (12,11) (13,1) (14,2) (15,9)
slot 7 (1,3) (2,8) (4,14) (5,0) (6,13) (9,7) (11,10) (12,15)
slot 8 (2,3) (5,9) (7,0) (8,11) (10,12) (13,4) (14,6) (15,1)
slot 9 (0,9) (1,7) (3,13) (4,15) (6,5) (10,8) (11,2) (12,14)
slot 10 (2,0) (3,4) (5,10) (6,11) (7,15) (8,1) (9,12) (14,13)
slot 11 (0,6) (1,9) (3,8) (4,12) (10,2) (11,14) (13,7) (15,5)
slot 12 (1,4) (2,5) (3,15) (7,11) (8,13) (9,6) (12,0) (14,10)
slot 13 (0,14) (2,7) (4,8) (5,12) (10,1) (11,3) (13,9) (15,6)
slot 14 (1,11) (2,13) (3,0) (4,5) (7,10) (8,15) (9,14) (12,6)
slot 15 (0,4) (5,1) (6,15) (8,12) (10,3) (11,7) (13,2) (14,9)
slot 16 (1,0) (2,11) (3,5) (4,6) (7,14) (9,8) (12,13) (15,10)
slot 17 (0,2) (5,7) (6,3) (8,4) (10,9) (12,1) (13,15) (14,11)
slot 18 (2,10) (3,1) (4,13) (7,6) (9,0) (11,8) (12,5) (15,14)
slot 19 (0,12) (5,2) (6,1) (8,3) (9,15) (10,7) (13,11) (14,4)
slot 20 (1,8) (3,2) (7,9) (11,6) (12,10) (13,0) (14,5) (15,4)
slot 21 (0,3) (2,12) (4,7) (6,14) (8,5) (9,1) (10,13) (15,11)
slot 22 (1,10) (2,15) (5,6) (7,8) (11,4) (12,9) (13,3) (14,0)
slot 23 (0,11) (3,7) (4,1) (6,9) (8,2) (10,5) (13,14) (15,12)
slot 24 (1,13) (5,11) (7,2) (8,6) (9,4) (10,14) (12,3) (15,0)
slot 25 (0,10) (2,1) (4,3) (6,12) (11,9) (13,5) (14,8) (15,7)
slot 26 (1,14) (2,9) (3,11) (5,15) (7,13) (8,0) (10,6) (12,4)
slot 27 (0,5) (4,10) (6,2) (7,1) (9,13) (11,12) (14,3) (15,8)
slot 28 (1,15) (2,14) (3,9) (5,4) (6,0) (10,11) (12,7) (13,8)
slot 29 (0,7) (4,2) (8,10) (9,5) (11,1) (13,6) (14,12) (15,3)

Πρόβλημα Middle 2

Το αποτέλεσμα που βρήκαμε είναι (35, 7422) το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους:

Ισχυροί περιορισμοί
246 = 216 ικανοποιούνται + 30 παραβιάζονται
CA1: 11

CA2: 2

CA3: 2

CA4: 19

BR1: 1

Ασθενείς περιορισμοί

CA1: 20

CA2: 2365

CA3: 1030

CA4: 2760

GA1: 57

SE1: 1190

και ωρολόγιο πρόγραμμα:

slot 0 (0,5) (1,14) (2,8) (4,6) (7,15) (9,11) (10,12) (13,3)
slot 1 (3,7) (6,5) (8,1) (10,9) (11,4) (12,13) (14,2) (15,0)
slot 2 (1,6) (4,10) (5,8) (7,0) (9,2) (12,11) (13,15) (14,3)
slot 3 (0,14) (2,5) (6,12) (8,4) (9,13) (10,7) (11,1) (15,3)
slot 4 (1,0) (3,2) (5,11) (6,15) (7,9) (12,8) (13,4) (14,10)
slot 5 (0,11) (2,4) (5,3) (7,6) (8,13) (9,14) (10,1) (15,12)
slot 6 (1,5) (3,8) (4,15) (6,9) (10,0) (11,7) (12,14) (13,2)
slot 7 (0,12) (2,10) (3,1) (5,15) (8,6) (9,4) (13,11) (14,7)
slot 8 (0,3) (1,9) (4,12) (6,14) (7,5) (10,13) (11,2) (15,8)
slot 9 (2,6) (3,10) (4,14) (5,9) (8,0) (11,15) (12,1) (13,7)
slot 10 (0,6) (1,4) (3,12) (5,13) (7,2) (9,15) (10,11) (14,8)
slot 11 (2,1) (4,7) (6,3) (9,0) (11,8) (12,5) (13,14) (15,10)
slot 12 (0,13) (1,7) (3,4) (5,10) (6,11) (8,9) (12,2) (14,15)
slot 13 (2,0) (4,5) (7,12) (9,3) (10,8) (11,14) (13,6) (15,1)
slot 14 (0,4) (1,13) (2,15) (3,11) (6,10) (8,7) (12,9) (14,5)
slot 15 (0,10) (3,6) (4,1) (5,12) (7,14) (9,8) (11,13) (15,2)
slot 16 (0,1) (2,9) (6,4) (7,3) (10,15) (11,5) (13,8) (14,12)
slot 17 (1,10) (3,13) (4,2) (5,0) (8,14) (9,6) (12,7) (15,11)
slot 18 (0,8) (2,12) (7,4) (10,3) (11,6) (13,9) (14,1) (15,5)
slot 19 (1,11) (3,15) (4,13) (5,14) (6,2) (8,10) (9,7) (12,0)
slot 20 (0,9) (1,3) (2,7) (8,12) (10,6) (13,5) (14,11) (15,4)
slot 21 (3,14) (4,9) (5,2) (6,8) (7,1) (11,0) (12,10) (15,13)
slot 22 (0,7) (1,15) (2,3) (8,11) (9,5) (10,4) (13,12) (14,6)
slot 23 (2,11) (4,0) (5,1) (6,13) (7,8) (10,14) (12,3) (15,9)
slot 24 (0,15) (1,2) (3,9) (7,13) (8,5) (11,10) (12,6) (14,4)
slot 25 (4,8) (5,7) (6,0) (9,12) (10,2) (11,3) (13,1) (15,14)

slot 26 (2,13) (3,5) (6,1) (7,11) (8,15) (9,10) (12,4) (14,0)
slot 27 (0,2) (1,8) (4,3) (5,6) (11,12) (13,10) (14,9) (15,7)
slot 28 (2,14) (4,11) (6,7) (8,3) (9,1) (10,5) (12,15) (13,0)
slot 29 (1,12) (3,0) (5,4) (7,10) (8,2) (11,9) (14,13) (15,6)

Πρόβλημα Middle 3

Η λύση που βρήκαμε είναι (0, 11809) με ανάλυση κόστους:

CA2: 2405
CA3: 925
CA4: 2995
GA1: 74
BR1: 10
BR2: 1060
FA2: 50
SE1: 4290

και ωρολόγιο πρόγραμμα:

slot 0 (1,0) (2,10) (3,14) (7,15) (8,6) (9,11) (12,4) (13,5)
slot 1 (0,5) (7,4) (9,3) (10,2) (11,1) (12,8) (13,6) (15,14)
slot 2 (0,9) (1,15) (3,7) (4,2) (5,11) (6,13) (8,10) (14,12)
slot 3 (1,11) (2,14) (3,4) (6,5) (7,12) (8,0) (10,9) (13,15)
slot 4 (0,7) (4,8) (5,13) (9,10) (11,2) (12,1) (14,3) (15,6)
slot 5 (2,9) (3,0) (6,8) (7,13) (10,5) (12,11) (14,4) (15,1)
slot 6 (1,12) (2,3) (4,11) (6,10) (7,0) (8,15) (9,5) (13,14)
slot 7 (0,13) (1,2) (4,3) (5,7) (9,6) (10,15) (11,8) (12,14)
slot 8 (0,8) (2,7) (3,11) (6,12) (10,4) (13,1) (14,5) (15,9)
slot 9 (1,10) (4,13) (5,3) (6,0) (7,14) (8,2) (11,15) (12,9)
slot 10 (0,1) (3,12) (4,6) (5,8) (9,15) (10,7) (13,2) (14,11)
slot 11 (0,14) (1,9) (2,12) (6,3) (7,10) (8,4) (11,13) (15,5)
slot 12 (3,6) (4,0) (5,2) (8,1) (9,14) (10,13) (11,7) (12,15)
slot 13 (0,12) (3,5) (6,2) (7,11) (10,1) (13,9) (14,8) (15,4)
slot 14 (0,15) (1,7) (2,13) (5,6) (8,3) (9,4) (11,14) (12,10)
slot 15 (1,6) (3,9) (4,5) (8,11) (10,12) (13,0) (14,7) (15,2)
slot 16 (2,1) (5,0) (6,14) (7,3) (9,8) (11,4) (13,12) (15,10)
slot 17 (0,11) (2,8) (4,15) (7,5) (9,13) (10,3) (12,6) (14,1)
slot 18 (1,13) (3,2) (4,12) (5,10) (6,7) (8,14) (11,9) (15,0)
slot 19 (0,3) (5,15) (7,1) (8,12) (9,2) (10,11) (13,4) (14,6)

slot 20 (2,11) (3,8) (4,9) (6,1) (10,0) (12,5) (14,13) (15,7)
slot 21 (1,14) (2,15) (4,10) (5,12) (7,6) (8,13) (9,0) (11,3)
slot 22 (0,4) (3,10) (5,1) (6,9) (12,7) (13,8) (14,2) (15,11)
slot 23 (1,4) (7,2) (8,9) (10,6) (11,5) (12,13) (14,0) (15,3)
slot 24 (0,10) (1,8) (2,5) (4,14) (6,15) (9,7) (11,12) (13,3)
slot 25 (0,6) (3,15) (4,1) (5,14) (7,9) (10,8) (12,2) (13,11)
slot 26 (1,5) (2,0) (3,13) (6,4) (8,7) (11,10) (14,9) (15,12)
slot 27 (2,6) (4,7) (8,5) (9,1) (11,0) (12,3) (13,10) (14,15)
slot 28 (0,2) (3,1) (5,4) (6,11) (7,8) (9,12) (10,14) (15,13)
slot 29 (1,3) (2,4) (5,9) (11,6) (12,0) (13,7) (14,10) (15,8)

Πρόβλημα Middle 4

Η λύση που βρήκαμε είναι (0, 7) με ανάλυση κόστους:

GA1: 7

και ωρολόγιο πρόγραμμα:

slot 0 (3,1) (5,2) (6,15) (7,4) (9,13) (10,11) (12,16) (14,0) (17,8)
slot 1 (1,4) (2,9) (3,7) (8,0) (10,6) (11,12) (13,14) (15,17) (16,5)
slot 2 (0,9) (1,7) (4,8) (5,11) (6,12) (10,2) (14,15) (16,13) (17,3)
slot 3 (0,15) (3,2) (7,16) (9,5) (11,4) (12,8) (13,10) (14,1) (17,6)
slot 4 (2,17) (5,0) (6,3) (7,9) (8,14) (10,4) (11,16) (13,12) (15,1)
slot 5 (0,2) (1,8) (3,15) (4,13) (5,6) (7,14) (9,12) (16,10) (17,11)
slot 6 (1,17) (2,13) (3,5) (6,9) (8,16) (11,7) (12,0) (14,10) (15,4)
slot 7 (4,6) (5,1) (7,0) (10,3) (11,2) (13,8) (14,12) (15,16) (17,9)
slot 8 (0,1) (2,14) (5,17) (8,6) (9,11) (10,7) (12,3) (13,15) (16,4)
slot 9 (1,13) (3,11) (4,5) (6,0) (7,8) (9,10) (12,15) (14,17) (16,2)
slot 10 (1,12) (2,7) (4,14) (5,10) (9,16) (11,6) (13,3) (15,8) (17,0)
slot 11 (0,4) (3,9) (6,2) (7,15) (8,5) (10,1) (11,13) (14,16) (17,12)
slot 12 (0,10) (2,8) (5,14) (9,1) (12,4) (13,6) (15,11) (16,3) (17,7)
slot 13 (0,11) (1,6) (3,14) (4,2) (7,12) (9,8) (10,15) (13,5) (16,17)
slot 14 (0,3) (2,15) (4,17) (5,12) (6,16) (8,10) (11,1) (13,7) (14,9)

slot 15 (1,2) (3,8) (4,9) (6,7) (11,14) (12,10) (15,5) (16,0) (17,13)

slot 16 (2,12) (4,3) (7,5) (8,11) (9,15) (10,17) (13,0) (14,6) (16,1)

slot 17 (0,14) (1,9) (3,17) (4,16) (5,7) (6,11) (8,2) (10,13) (15,12)

slot 18 (0,6) (2,1) (3,13) (8,15) (11,5) (12,7) (14,4) (16,9) (17,10)

slot 19 (2,3) (5,4) (7,6) (8,1) (9,0) (10,12) (13,17) (15,14) (16,11)

slot 20 (0,13) (1,10) (2,11) (4,7) (6,5) (8,9) (12,14) (15,3) (17,16)

slot 21 (0,17) (3,10) (4,15) (9,2) (11,8) (12,6) (13,1) (14,5) (16,7)

slot 22 (1,14) (2,16) (5,13) (6,4) (7,3) (8,17) (10,9) (12,11) (15,0)

slot 23 (1,3) (4,12) (5,9) (7,10) (8,13) (11,0) (15,6) (16,14) (17,2)

slot 24 (0,5) (1,15) (3,6) (8,4) (9,17) (11,10) (12,2) (13,16) (14,7)

slot 25 (2,0) (3,12) (4,1) (5,16) (6,14) (10,8) (11,17) (13,9) (15,7)

slot 26 (1,0) (4,10) (7,2) (8,3) (9,6) (14,11) (15,13) (16,12) (17,5)

slot 27 (0,8) (5,15) (7,11) (9,3) (10,14) (12,1) (13,2) (16,6) (17,4)

slot 28 (1,5) (2,10) (3,16) (4,0) (6,17) (8,7) (11,15) (12,9) (14,13)

slot 29 (0,12) (2,4) (5,8) (6,1) (7,17) (10,16) (13,11) (14,3) (15,9)

slot 30 (6,13) (7,1) (9,4) (10,0) (11,3) (12,5) (15,2) (16,8) (17,14)

slot 31 (1,16) (2,6) (3,0) (4,11) (7,13) (8,12) (9,14) (10,5) (17,15)

slot 32 (0,16) (1,11) (5,3) (6,8) (9,7) (12,17) (13,4) (14,2) (15,10)

slot 33 (0,7) (2,5) (3,4) (6,10) (11,9) (12,13) (14,8) (16,15) (17,1)

Η λύση αυτή έχει την ίδια τιμή αντικειμενικής συνάρτησης με την βέλτιστη λύση που είναι δημοσιευμένη από τον διαγωνισμό ITC2021, αλλά το δικό μας ωρολόγιο πρόγραμμα είναι **διαφορετικό**. Η απόσταση d (σχέση 55) ανάμεσα στην δική μας λύση και την λύση του διαγωνισμού ισούται με $d = 0.8497$, γεγονός που σημαίνει ότι πρόκειται για πολύ διαφορετική λύση.

Πρόβλημα Middle 5

Η λύση που βρήκαμε είναι (0, 1029) με ανάλυση κόστους:

CA1: 21

CA2: 30

GA1: 58

BR2: 920

και ωρολόγιο πρόγραμμα:

slot 0 (1,14) (2,10) (7,9) (8,0) (11,6) (12,5) (15,3) (16,4) (17,13)
slot 1 (0,12) (3,17) (4,2) (5,8) (6,16) (9,15) (10,7) (13,1) (14,11)
slot 2 (0,6) (3,2) (4,7) (9,10) (11,1) (12,8) (15,13) (16,14) (17,5)
slot 3 (2,12) (5,1) (6,15) (7,16) (8,17) (10,3) (11,9) (13,4) (14,0)
slot 4 (1,6) (2,11) (4,14) (7,17) (9,5) (12,3) (13,0) (15,8) (16,10)
slot 5 (2,1) (3,16) (5,15) (6,4) (8,13) (9,0) (11,7) (14,10) (17,12)
slot 6 (0,17) (1,12) (3,8) (4,9) (7,14) (10,6) (13,5) (15,2) (16,11)
slot 7 (0,4) (1,3) (5,16) (6,2) (7,15) (10,12) (11,8) (14,13) (17,9)
slot 8 (0,2) (4,1) (5,7) (8,16) (9,3) (12,14) (13,6) (15,10) (17,11)
slot 9 (1,9) (2,13) (3,4) (6,17) (7,0) (10,8) (11,12) (14,5) (16,15)
slot 10 (0,1) (3,14) (4,5) (8,2) (9,6) (12,16) (13,7) (15,11) (17,10)
slot 11 (1,8) (5,0) (6,12) (7,3) (10,13) (11,4) (14,2) (15,17) (16,9)
slot 12 (0,16) (1,17) (2,7) (3,5) (4,10) (8,6) (9,14) (12,15) (13,11)
slot 13 (3,13) (7,6) (8,14) (10,5) (11,0) (12,9) (15,1) (16,2) (17,4)
slot 14 (0,15) (1,10) (2,9) (5,11) (6,3) (8,7) (12,4) (13,16) (14,17)
slot 15 (3,11) (4,8) (5,6) (7,12) (9,13) (10,0) (15,14) (16,1) (17,2)
slot 16 (0,3) (1,7) (2,5) (4,15) (6,14) (8,9) (11,10) (13,12) (17,16)
slot 17 (3,0) (5,13) (6,11) (7,4) (9,17) (10,1) (12,2) (14,15) (16,8)
slot 18 (0,9) (2,14) (4,12) (8,1) (11,5) (13,10) (15,6) (16,7) (17,3)
slot 19 (1,11) (3,7) (4,0) (5,17) (9,2) (10,16) (12,6) (13,15) (14,8)
slot 20 (0,13) (2,3) (6,1) (7,10) (8,5) (11,17) (14,4) (15,9) (16,12)
slot 21 (1,13) (2,0) (3,6) (4,11) (5,14) (8,15) (9,16) (10,17) (12,7)
slot 22 (0,8) (6,9) (7,1) (10,4) (11,3) (13,2) (14,12) (16,5) (17,15)
slot 23 (1,0) (2,17) (3,10) (4,13) (6,5) (9,8) (12,11) (14,16) (15,7)
slot 24 (0,14) (1,4) (2,15) (5,10) (7,11) (9,12) (13,3) (16,6) (17,8)
slot 25 (3,12) (5,2) (6,7) (8,4) (10,9) (11,13) (14,1) (15,16) (17,0)
slot 26 (0,5) (1,15) (2,16) (4,3) (7,8) (9,11) (12,10) (13,14) (17,6)
slot 27 (3,1) (5,9) (6,13) (8,11) (10,2) (12,17) (14,7) (15,4) (16,0)
slot 28 (0,10) (1,16) (4,6) (5,12) (7,2) (8,3) (11,15) (13,9) (17,14)

slot 29 (0,7) (2,4) (6,8) (9,1) (10,11) (12,13) (14,3) (15,5) (16,17)
slot 30 (1,2) (3,9) (4,16) (6,0) (7,5) (8,10) (11,14) (13,17) (15,12)
slot 31 (0,11) (2,8) (5,3) (9,4) (10,15) (12,1) (14,6) (16,13) (17,7)
slot 32 (1,5) (4,17) (6,10) (7,13) (8,12) (11,2) (14,9) (15,0) (16,3)
slot 33 (2,6) (3,15) (5,4) (9,7) (10,14) (11,16) (12,0) (13,8) (17,1)

Πρόβλημα Middle 6

Η λύση που βρήκαμε είναι (0, 1920) με ανάλυση κόστους:

CA3: 490
CA4: 135
BR1: 15
BR2: 1120
SE1: 160

και ωρολόγιο πρόγραμμα:

slot 0 (1,13) (3,4) (5,7) (8,15) (9,17) (10,2) (11,0) (14,12) (16,6)
slot 1 (0,1) (4,9) (5,8) (6,3) (7,16) (12,10) (13,2) (14,17) (15,11)
slot 2 (1,12) (2,7) (3,11) (4,6) (9,5) (13,8) (15,14) (16,0) (17,10)
slot 3 (0,13) (5,2) (6,17) (7,3) (8,1) (10,16) (11,9) (12,15) (14,4)
slot 4 (1,7) (2,11) (4,10) (5,3) (6,14) (9,0) (13,15) (16,8) (17,12)
slot 5 (2,17) (3,12) (7,4) (8,9) (10,13) (11,5) (14,0) (15,6) (16,1)
slot 6 (0,4) (1,6) (5,16) (8,2) (9,15) (10,3) (12,11) (13,14) (17,7)
slot 7 (1,15) (2,0) (3,17) (4,8) (5,10) (6,13) (7,14) (12,9) (16,11)
slot 8 (0,7) (6,2) (10,8) (11,1) (13,9) (14,5) (15,3) (16,12) (17,4)
slot 9 (2,15) (3,16) (4,11) (7,13) (8,6) (9,14) (10,0) (12,5) (17,1)
slot 10 (0,12) (1,10) (2,14) (3,8) (4,13) (5,15) (6,7) (11,17) (16,9)
slot 11 (0,3) (5,6) (7,8) (9,2) (12,4) (13,11) (14,1) (15,10) (17,16)
slot 12 (2,1) (3,14) (4,5) (6,0) (8,17) (10,9) (11,7) (12,13) (15,16)
slot 13 (1,3) (2,4) (5,13) (7,15) (8,12) (9,6) (10,11) (16,14) (17,0)
slot 14 (0,5) (3,2) (4,1) (6,12) (7,9) (11,8) (13,16) (14,10) (15,17)
slot 15 (2,16) (4,15) (5,1) (8,0) (9,3) (10,6) (11,14) (12,7) (13,17)
slot 16 (0,15) (1,9) (3,13) (6,11) (7,10) (12,2) (14,8) (16,4) (17,5)
slot 17 (0,14) (2,8) (4,3) (5,9) (6,16) (7,17) (11,12) (13,10) (15,1)
slot 18 (1,0) (8,4) (9,12) (10,5) (11,3) (14,6) (15,13) (16,7) (17,2)
slot 19 (0,2) (3,10) (4,14) (6,15) (7,5) (9,8) (12,16) (13,1) (17,11)
slot 20 (1,17) (2,6) (5,4) (7,0) (8,10) (11,13) (14,9) (15,12) (16,3)
slot 21 (0,16) (1,8) (3,5) (4,7) (9,13) (10,15) (11,2) (12,14) (17,6)

slot 22 (2,9) (3,15) (6,5) (8,7) (10,1) (12,0) (13,4) (14,11) (16,17)
slot 23 (4,12) (5,0) (7,1) (9,10) (11,16) (13,6) (14,3) (15,2) (17,8)
slot 24 (0,9) (1,14) (2,5) (3,7) (8,13) (10,17) (11,4) (12,6) (16,15)
slot 25 (4,2) (6,1) (8,11) (9,7) (10,12) (13,0) (14,16) (15,5) (17,3)
slot 26 (0,8) (1,16) (2,13) (3,9) (5,14) (7,6) (11,10) (12,17) (15,4)
slot 27 (1,5) (4,0) (6,10) (7,12) (8,3) (9,11) (14,2) (16,13) (17,15)
slot 28 (0,6) (2,12) (3,1) (5,11) (9,16) (10,4) (13,7) (15,8) (17,14)
slot 29 (0,10) (4,17) (6,9) (8,5) (11,15) (12,1) (13,3) (14,7) (16,2)
slot 30 (1,11) (3,6) (5,12) (7,2) (8,16) (9,4) (10,14) (15,0) (17,13)
slot 31 (0,17) (2,3) (6,4) (7,11) (9,1) (12,8) (13,5) (14,15) (16,10)
slot 32 (1,2) (3,0) (4,16) (5,17) (8,14) (10,7) (11,6) (13,12) (15,9)
slot 33 (0,11) (1,4) (2,10) (6,8) (12,3) (14,13) (15,7) (16,5) (17,9)

Πρόβλημα Middle 7

Η λύση που βρήκαμε είναι (0, 2672) με ανάλυση κόστους:

CA1: 19
CA2: 870
CA4: 335
GA1: 13
BR1: 5
BR2: 1240
SE1: 190

και ωρολόγιο πρόγραμμα:

slot 0 (0,4) (6,7) (9,3) (10,1) (12,2) (14,5) (15,11) (16,8) (17,13)
slot 1 (0,15) (1,17) (2,16) (3,14) (4,11) (5,10) (8,6) (9,13) (12,7)
slot 2 (1,4) (3,6) (7,14) (8,5) (10,9) (13,0) (15,2) (16,11) (17,12)
slot 3 (2,9) (4,3) (5,15) (6,13) (7,8) (11,1) (12,10) (14,0) (16,17)
slot 4 (0,11) (1,7) (3,12) (9,5) (10,6) (13,16) (14,2) (15,8) (17,4)
slot 5 (2,7) (3,10) (8,14) (9,15) (11,5) (12,6) (13,1) (16,4) (17,0)
slot 6 (0,16) (1,8) (4,12) (5,3) (6,9) (7,11) (10,2) (14,17) (15,13)
slot 7 (3,8) (5,4) (6,0) (11,2) (12,9) (13,10) (14,1) (15,16) (17,7)
slot 8 (0,12) (2,5) (4,6) (7,13) (8,17) (9,1) (10,15) (11,14) (16,3)
slot 9 (1,16) (3,11) (4,10) (5,6) (9,0) (12,8) (13,14) (15,7) (17,2)
slot 10 (0,7) (1,12) (2,3) (8,9) (11,10) (13,4) (14,6) (16,5) (17,15)
slot 11 (3,17) (4,2) (5,0) (6,16) (7,9) (10,8) (11,13) (14,12) (15,1)
slot 12 (1,2) (7,4) (8,0) (9,11) (10,14) (12,16) (13,3) (15,6) (17,5)

slot 13 (0,1) (3,7) (4,15) (5,12) (6,17) (9,14) (11,8) (13,2) (16,10)
slot 14 (1,6) (3,0) (4,9) (5,13) (7,10) (8,2) (12,15) (14,16) (17,11)
slot 15 (1,5) (2,6) (7,16) (9,17) (10,0) (11,12) (13,8) (14,4) (15,3)
slot 16 (0,2) (3,1) (5,7) (6,11) (8,4) (12,13) (15,14) (16,9) (17,10)
slot 17 (2,15) (4,1) (6,5) (9,8) (10,12) (11,7) (13,17) (14,3) (16,0)
slot 18 (0,13) (1,14) (2,11) (3,9) (7,6) (8,15) (10,5) (12,4) (17,16)
slot 19 (4,17) (5,9) (6,10) (7,1) (8,3) (12,0) (13,15) (14,11) (16,2)
slot 20 (0,8) (2,12) (3,16) (9,6) (10,13) (11,4) (14,7) (15,5) (17,1)
slot 21 (1,9) (2,14) (5,11) (6,4) (7,17) (8,10) (12,3) (15,0) (16,13)
slot 22 (0,17) (4,5) (7,2) (9,16) (10,3) (11,15) (12,1) (13,6) (14,8)
slot 23 (0,9) (1,11) (2,4) (3,13) (5,14) (6,12) (10,7) (16,15) (17,8)
slot 24 (2,17) (4,16) (6,15) (7,3) (8,1) (9,10) (11,0) (12,14) (13,5)
slot 25 (0,6) (3,2) (5,8) (10,4) (11,17) (13,7) (14,9) (15,12) (16,1)
slot 26 (1,15) (2,10) (4,13) (5,16) (6,14) (7,0) (8,11) (9,12) (17,3)
slot 27 (0,5) (1,13) (8,7) (9,2) (11,3) (12,17) (14,10) (15,4) (16,6)
slot 28 (2,8) (3,5) (4,0) (6,1) (7,15) (10,16) (11,9) (13,12) (17,14)
slot 29 (0,14) (1,3) (5,17) (6,2) (8,12) (9,4) (13,11) (15,10) (16,7)
slot 30 (1,10) (2,0) (3,15) (4,7) (8,16) (11,6) (12,5) (14,13) (17,9)
slot 31 (0,10) (2,1) (4,8) (6,3) (7,5) (12,11) (13,9) (15,17) (16,14)
slot 32 (1,0) (3,4) (5,2) (8,13) (9,7) (10,11) (14,15) (16,12) (17,6)
slot 33 (0,3) (2,13) (4,14) (5,1) (6,8) (7,12) (10,17) (11,16) (15,9)

Πρόβλημα Middle 8

Η λύση που βρήκαμε είναι (0, 447) με ανάλυση κόστους:

CA2: 30
CA3: 275
CA4: 125
GA1: 17

και ωρολόγιο πρόγραμμα:

slot 0 (0,3) (4,1) (5,16) (6,13) (8,9) (10,12) (11,15) (14,2) (17,7)
slot 1 (1,14) (2,9) (5,4) (6,8) (7,3) (13,12) (15,0) (16,11) (17,10)
slot 2 (2,6) (3,17) (4,7) (8,15) (9,16) (10,13) (11,1) (12,5) (14,0)
slot 3 (0,5) (1,6) (7,17) (8,3) (9,11) (10,2) (13,16) (14,12) (15,4)
slot 4 (0,9) (1,2) (3,13) (4,10) (5,11) (6,14) (15,7) (16,12) (17,8)
slot 5 (2,16) (7,1) (8,0) (9,17) (10,3) (11,4) (12,6) (13,15) (14,5)
slot 6 (0,8) (1,7) (3,9) (4,13) (5,12) (6,16) (11,2) (15,10) (17,14)

slot 7 (3,6) (7,4) (8,5) (10,11) (12,0) (13,2) (14,9) (15,1) (16,17)
slot 8 (0,12) (1,11) (2,5) (4,8) (9,7) (10,6) (14,3) (16,13) (17,15)
slot 9 (1,16) (3,15) (5,10) (6,4) (7,0) (9,12) (11,8) (13,14) (17,2)
slot 10 (0,13) (2,3) (5,1) (8,6) (10,7) (11,9) (12,17) (14,4) (15,16)
slot 11 (1,17) (4,0) (6,3) (7,2) (9,15) (12,10) (13,8) (14,11) (16,5)
slot 12 (1,8) (2,0) (3,4) (9,14) (10,16) (11,12) (13,7) (15,5) (17,6)
slot 13 (0,17) (2,10) (4,16) (5,3) (6,11) (7,13) (8,14) (12,1) (15,9)
slot 14 (5,9) (6,7) (8,1) (11,10) (12,2) (13,0) (14,15) (16,3) (17,4)
slot 15 (1,5) (2,4) (3,0) (7,12) (9,13) (10,14) (15,8) (16,6) (17,11)
slot 16 (0,16) (2,1) (4,12) (5,6) (8,17) (9,3) (13,10) (14,7) (15,11)
slot 17 (0,14) (3,7) (4,5) (6,9) (10,15) (11,13) (12,8) (16,2) (17,1)
slot 18 (1,12) (5,13) (7,10) (8,2) (9,6) (11,0) (14,17) (15,3) (16,4)
slot 19 (2,15) (3,11) (4,14) (5,17) (8,16) (9,0) (10,1) (12,7) (13,6)
slot 20 (0,7) (1,15) (4,9) (6,2) (11,5) (13,3) (14,8) (16,10) (17,12)
slot 21 (0,4) (1,3) (5,2) (7,11) (8,13) (10,17) (12,9) (14,16) (15,6)
slot 22 (2,7) (3,12) (6,5) (9,1) (10,8) (11,14) (13,4) (15,17) (16,0)
slot 23 (0,10) (4,2) (5,15) (8,7) (11,3) (12,13) (14,6) (16,1) (17,9)
slot 24 (1,10) (2,8) (3,14) (5,0) (6,15) (7,16) (9,4) (12,11) (13,17)
slot 25 (1,13) (2,17) (3,8) (4,11) (6,0) (7,14) (10,5) (15,12) (16,9)
slot 26 (0,15) (8,10) (9,2) (11,6) (12,4) (13,5) (14,1) (16,7) (17,3)
slot 27 (2,14) (3,1) (4,17) (5,8) (6,12) (7,9) (10,0) (11,16) (15,13)
slot 28 (0,6) (1,9) (2,12) (3,10) (5,14) (7,15) (8,4) (13,11) (17,16)
slot 29 (0,2) (4,3) (6,17) (9,5) (11,7) (12,15) (13,1) (14,10) (16,8)
slot 30 (1,4) (2,13) (3,5) (7,6) (8,11) (10,9) (12,16) (15,14) (17,0)
slot 31 (0,1) (3,2) (4,6) (5,7) (8,12) (9,10) (11,17) (14,13) (16,15)
slot 32 (0,11) (6,1) (7,8) (10,4) (12,3) (13,9) (15,2) (16,14) (17,5)
slot 33 (1,0) (2,11) (3,16) (4,15) (6,10) (7,5) (9,8) (12,14) (17,13)

Πρόβλημα Middle 9

Η λύση που βρήκαμε είναι (0, 2185) με ανάλυση κόστους:

CA2: 85
CA3: 515
CA4: 225
BR1: 20
BR2: 1340

και ωρολόγιο πρόγραμμα:

slot 0 (0,4) (2,6) (9,12) (10,5) (11,15) (13,3) (14,8) (16,1) (17,7)
slot 1 (1,10) (3,11) (4,7) (5,0) (6,16) (8,9) (12,13) (14,17) (15,2)
slot 2 (0,6) (4,14) (7,1) (8,5) (10,12) (11,2) (13,17) (15,9) (16,3)
slot 3 (0,8) (2,14) (3,7) (6,1) (9,4) (10,16) (12,11) (13,15) (17,5)
slot 4 (1,14) (3,12) (4,10) (5,6) (7,11) (8,17) (9,13) (15,0) (16,2)
slot 5 (2,10) (6,3) (7,0) (11,8) (12,1) (13,5) (14,16) (15,4) (17,9)
slot 6 (0,12) (1,13) (3,14) (5,15) (8,16) (9,7) (10,6) (11,4) (17,2)
slot 7 (1,4) (3,9) (5,16) (6,17) (7,10) (12,2) (13,0) (14,11) (15,8)
slot 8 (0,17) (2,3) (4,5) (7,12) (8,10) (9,6) (11,1) (14,15) (16,13)
slot 9 (0,10) (1,8) (5,3) (9,2) (11,6) (12,16) (13,14) (15,7) (17,4)
slot 10 (1,9) (2,13) (3,17) (6,15) (7,5) (8,4) (10,11) (12,14) (16,0)
slot 11 (0,9) (3,1) (4,6) (5,12) (7,2) (8,13) (14,10) (15,16) (17,11)
slot 12 (1,0) (2,4) (6,8) (9,11) (10,3) (12,15) (13,7) (14,5) (16,17)
slot 13 (2,0) (3,8) (4,12) (5,11) (7,14) (13,6) (15,1) (16,9) (17,10)
slot 14 (0,3) (1,5) (4,16) (6,7) (8,2) (10,15) (11,13) (12,17) (14,9)
slot 15 (0,14) (2,1) (3,4) (6,12) (7,8) (9,5) (13,10) (16,11) (17,15)
slot 16 (4,13) (5,2) (8,12) (10,9) (11,0) (14,6) (15,3) (16,7) (17,1)
slot 17 (2,17) (3,0) (5,14) (6,11) (8,7) (9,16) (10,4) (13,1) (15,12)
slot 18 (0,5) (1,3) (4,15) (7,16) (9,8) (11,10) (12,6) (14,2) (17,13)
slot 19 (1,7) (2,8) (4,9) (5,17) (10,13) (11,3) (12,0) (15,14) (16,6)
slot 20 (0,1) (2,11) (3,10) (5,4) (8,6) (9,15) (13,16) (14,7) (17,12)
slot 21 (4,1) (6,2) (7,9) (10,0) (11,14) (12,3) (13,8) (15,17) (16,5)
slot 22 (1,16) (2,15) (5,7) (6,4) (8,0) (9,3) (10,17) (11,12) (14,13)
slot 23 (0,15) (1,11) (3,13) (4,2) (5,8) (9,10) (12,7) (16,14) (17,6)
slot 24 (0,16) (2,12) (6,9) (7,13) (8,3) (10,1) (11,17) (14,4) (15,5)
slot 25 (3,6) (4,11) (5,1) (7,15) (9,14) (12,10) (13,2) (16,8) (17,0)
slot 26 (0,13) (2,9) (3,16) (4,17) (6,14) (8,1) (11,7) (12,5) (15,10)
slot 27 (1,12) (5,9) (6,0) (7,3) (8,15) (10,2) (13,11) (16,4) (17,14)
slot 28 (1,17) (2,5) (3,15) (7,6) (10,8) (11,16) (12,9) (13,4) (14,0)
slot 29 (2,7) (3,5) (4,0) (6,13) (9,17) (12,8) (14,1) (15,11) (16,10)
slot 30 (0,2) (1,6) (5,10) (7,4) (8,14) (11,9) (13,12) (16,15) (17,3)
slot 31 (0,11) (3,2) (4,8) (6,5) (9,1) (10,7) (14,12) (15,13) (17,16)
slot 32 (1,15) (2,16) (5,13) (6,10) (7,17) (8,11) (9,0) (12,4) (14,3)
slot 33 (0,7) (1,2) (4,3) (10,14) (11,5) (13,9) (15,6) (16,12) (17,8)

Πρόβλημα Middle 10

Η λύση που βρήκαμε είναι (0, 2211) με ανάλυση κόστους:

CA1: 21

CA2: 645

CA4: 1510

GA1: 35

και ωρολόγιο πρόγραμμα:

slot 0 (0,1) (3,17) (8,16) (9,6) (10,7) (12,13) (14,5) (15,11) (18,4) (19,2)
slot 1 (1,9) (2,4) (3,19) (5,18) (6,12) (7,14) (11,10) (13,8) (16,15) (17,0)
slot 2 (0,3) (4,10) (5,11) (6,17) (8,1) (9,2) (12,16) (14,15) (18,13) (19,7)
slot 3 (0,12) (1,14) (2,18) (3,9) (7,4) (10,19) (11,6) (15,13) (16,5) (17,8)
slot 4 (2,10) (4,11) (6,16) (8,18) (9,0) (12,5) (13,3) (14,17) (15,7) (19,1)
slot 5 (0,13) (3,4) (5,19) (7,6) (8,15) (10,1) (11,2) (16,14) (17,12) (18,9)
slot 6 (1,12) (2,5) (4,0) (6,15) (9,7) (13,11) (14,8) (16,10) (18,3) (19,17)
slot 7 (0,6) (3,14) (5,1) (8,4) (10,9) (11,16) (12,19) (13,7) (15,2) (17,18)
slot 8 (0,19) (1,6) (2,8) (4,17) (5,3) (7,12) (10,15) (11,18) (14,13) (16,9)
slot 9 (3,6) (8,7) (9,5) (11,14) (12,10) (13,2) (15,0) (17,1) (18,16) (19,4)
slot 10 (0,8) (1,15) (5,4) (7,11) (10,18) (12,3) (14,9) (16,2) (17,13) (19,6)
slot 11 (2,14) (3,7) (4,6) (5,17) (9,8) (10,0) (11,1) (13,16) (15,12) (18,19)
slot 12 (0,16) (1,18) (6,10) (7,2) (8,3) (12,9) (13,5) (14,4) (17,11) (19,15)
slot 13 (2,3) (4,1) (6,13) (7,0) (9,17) (10,8) (11,12) (15,5) (16,19) (18,14)
slot 14 (0,2) (1,7) (3,15) (5,10) (8,11) (13,4) (14,12) (17,16) (18,6) (19,9)
slot 15 (1,13) (4,9) (6,14) (7,5) (10,17) (11,0) (12,2) (15,18) (16,3) (19,8)
slot 16 (2,1) (3,11) (4,16) (5,0) (8,6) (9,15) (13,10) (14,19) (17,7) (18,12)
slot 17 (0,18) (1,3) (2,17) (6,5) (7,16) (10,14) (11,9) (12,8) (13,19) (15,4)
slot 18 (3,10) (4,12) (6,2) (8,5) (9,13) (14,0) (16,1) (17,15) (18,7) (19,11)
slot 19 (0,9) (1,16) (2,6) (5,13) (7,3) (8,17) (10,4) (11,19) (12,18) (15,14)
slot 20 (3,1) (4,15) (6,11) (9,10) (13,0) (14,7) (16,12) (17,2) (18,8) (19,5)
slot 21 (1,5) (2,0) (3,16) (7,15) (9,18) (10,13) (11,8) (12,4) (17,6) (19,14)
slot 22 (0,17) (4,19) (5,12) (6,7) (8,2) (13,1) (14,3) (15,9) (16,11) (18,10)
slot 23 (1,8) (2,16) (3,0) (7,13) (9,4) (10,6) (11,15) (12,14) (17,5) (19,18)
slot 24 (4,7) (5,2) (6,3) (8,10) (9,1) (13,12) (14,11) (15,19) (16,17) (18,0)
slot 25 (0,15) (1,4) (2,19) (3,18) (5,6) (7,10) (8,14) (12,11) (16,13) (17,9)
slot 26 (4,2) (6,8) (9,3) (10,16) (11,7) (12,0) (14,1) (15,17) (18,5) (19,13)
slot 27 (0,4) (1,10) (2,11) (3,12) (5,15) (7,19) (8,9) (13,18) (16,6) (17,14)
slot 28 (4,13) (5,16) (6,9) (10,3) (11,17) (12,7) (14,2) (15,8) (18,1) (19,0)

slot 29 (0,11) (1,19) (2,15) (3,8) (6,4) (7,17) (9,12) (10,5) (13,14) (16,18)
slot 30 (2,7) (4,18) (5,9) (8,0) (11,13) (12,1) (14,6) (15,10) (17,3) (19,16)
slot 31 (0,14) (1,17) (3,5) (6,19) (7,8) (9,11) (10,12) (13,15) (16,4) (18,2)
slot 32 (2,9) (4,5) (6,0) (7,18) (8,13) (11,3) (14,16) (15,1) (17,10) (19,12)
slot 33 (0,7) (1,11) (4,3) (5,14) (9,19) (10,2) (12,15) (13,6) (16,8) (18,17)
slot 34 (0,10) (2,12) (3,13) (6,1) (7,9) (8,19) (11,5) (14,18) (15,16) (17,4)
slot 35 (1,2) (4,8) (5,7) (9,14) (10,11) (12,6) (13,17) (16,0) (18,15) (19,3)
slot 36 (0,5) (2,13) (6,18) (7,1) (8,12) (9,16) (11,4) (14,10) (15,3) (17,19)
slot 37 (1,0) (3,2) (4,14) (5,8) (12,17) (13,9) (15,6) (16,7) (18,11) (19,10)

Πρόβλημα Middle 11

Η λύση που βρήκαμε είναι (0, 3638) με ανάλυση κόστους:

CA2: 1405
CA3: 360
CA4: 1865
GA1: 8

και ωρολόγιο πρόγραμμα:

slot 0 (0,17) (2,7) (4,3) (5,19) (9,6) (10,14) (11,1) (12,8) (15,18) (16,13)
slot 1 (1,12) (3,11) (6,4) (7,9) (8,5) (13,2) (15,16) (17,14) (18,0) (19,10)
slot 2 (1,3) (4,13) (6,18) (7,10) (9,2) (11,0) (12,17) (14,5) (16,8) (19,15)
slot 3 (0,3) (2,16) (5,15) (8,4) (9,1) (10,6) (12,13) (14,19) (17,11) (18,7)
slot 4 (0,12) (2,8) (3,19) (4,17) (5,16) (7,14) (11,9) (13,6) (15,1) (18,10)
slot 5 (1,7) (4,5) (6,3) (8,0) (9,13) (10,12) (15,11) (16,14) (17,2) (19,18)
slot 6 (0,13) (2,6) (5,3) (7,15) (11,10) (12,16) (14,9) (17,8) (18,1) (19,4)
slot 7 (0,1) (3,14) (5,11) (6,19) (8,7) (10,9) (12,2) (13,15) (16,4) (18,17)
slot 8 (1,5) (3,12) (4,18) (7,0) (9,8) (10,16) (11,19) (13,14) (15,2) (17,6)
slot 9 (0,10) (1,4) (2,5) (7,11) (8,13) (14,15) (16,6) (17,9) (18,3) (19,12)
slot 10 (3,2) (4,0) (5,7) (6,15) (8,10) (9,16) (11,14) (12,18) (13,17) (19,1)
slot 11 (0,19) (2,18) (4,12) (5,6) (10,1) (13,11) (14,8) (15,9) (16,3) (17,7)
slot 12 (1,13) (7,16) (8,6) (9,0) (10,4) (12,11) (14,2) (15,3) (18,5) (19,17)
slot 13 (0,14) (2,19) (3,10) (6,7) (8,1) (9,4) (11,16) (12,5) (17,15) (18,13)
slot 14 (1,17) (3,8) (4,2) (5,9) (11,6) (13,10) (14,18) (15,12) (16,0) (19,7)
slot 15 (0,15) (2,11) (6,1) (7,4) (8,18) (9,3) (10,5) (13,19) (14,12) (17,16)
slot 16 (1,2) (3,17) (5,0) (6,14) (7,13) (11,4) (12,9) (15,10) (18,16) (19,8)
slot 17 (0,2) (1,14) (3,7) (4,15) (8,11) (10,17) (12,6) (13,5) (16,19) (18,9)
slot 18 (2,10) (6,0) (7,12) (8,15) (9,19) (11,18) (13,3) (14,4) (16,1) (17,5)

slot 19 (0,9) (3,16) (4,1) (5,2) (6,11) (7,8) (10,18) (15,13) (17,12) (19,14)
slot 20 (1,8) (2,0) (9,5) (11,17) (12,10) (13,4) (14,6) (16,7) (18,15) (19,3)
slot 21 (1,19) (3,0) (4,14) (5,13) (7,6) (8,9) (10,15) (11,12) (16,2) (17,18)
slot 22 (0,16) (2,3) (4,8) (6,17) (9,18) (10,11) (12,7) (14,1) (15,5) (19,13)
slot 23 (0,5) (1,6) (2,12) (3,4) (9,11) (13,7) (15,14) (16,10) (17,19) (18,8)
slot 24 (1,0) (3,13) (4,9) (5,18) (6,10) (7,2) (8,17) (11,15) (12,19) (14,16)
slot 25 (0,4) (2,13) (9,7) (10,8) (14,3) (15,6) (16,11) (17,1) (18,12) (19,5)
slot 26 (1,10) (2,15) (3,18) (4,11) (5,14) (6,16) (7,17) (8,12) (13,9) (19,0)
slot 27 (0,18) (5,4) (6,2) (7,19) (9,10) (11,8) (12,1) (14,13) (16,15) (17,3)
slot 28 (1,9) (2,4) (8,14) (10,19) (11,5) (12,3) (13,0) (15,7) (16,17) (18,6)
slot 29 (0,8) (2,14) (3,1) (4,7) (5,12) (6,9) (10,13) (15,17) (18,11) (19,16)
slot 30 (0,11) (1,16) (6,12) (7,5) (8,3) (9,15) (13,18) (14,10) (17,4) (19,2)
slot 31 (2,17) (3,15) (4,6) (5,8) (10,0) (11,7) (12,14) (13,1) (16,9) (18,19)
slot 32 (1,18) (3,5) (6,13) (8,19) (9,17) (10,7) (11,2) (14,0) (15,4) (16,12)
slot 33 (0,6) (2,9) (4,16) (5,1) (7,3) (12,15) (13,8) (17,10) (18,14) (19,11)
slot 34 (1,11) (3,9) (5,17) (8,16) (10,2) (13,12) (14,7) (15,0) (18,4) (19,6)
slot 35 (2,1) (4,10) (6,8) (7,18) (9,14) (11,3) (12,0) (15,19) (16,5) (17,13)
slot 36 (0,7) (1,15) (6,5) (8,2) (10,3) (11,13) (12,4) (14,17) (16,18) (19,9)
slot 37 (3,6) (4,19) (5,10) (7,1) (9,12) (13,16) (14,11) (15,8) (17,0) (18,2)

Πρόβλημα Middle 12

Η λύση που βρήκαμε είναι (0, 3291) με ανάλυση κόστους:

CA1: 21
CA2: 260
CA3: 50
BR1: 30
BR2: 2060
FA2: 20
SE1: 850

και ωρολόγιο πρόγραμμα:

slot 0 (1,15) (2,0) (3,10) (7,16) (8,6) (11,5) (12,19) (14,4) (17,9) (18,13)
slot 1 (0,18) (2,8) (5,3) (10,9) (11,4) (13,6) (15,12) (16,14) (17,7) (19,1)
slot 2 (4,17) (5,0) (6,19) (7,2) (8,15) (9,1) (12,11) (14,3) (16,13) (18,10)
slot 3 (0,13) (1,5) (6,7) (9,3) (10,16) (12,8) (15,2) (17,11) (18,4) (19,14)
slot 4 (0,10) (2,12) (3,11) (4,6) (5,9) (7,18) (13,1) (14,15) (16,8) (19,17)
slot 5 (1,12) (2,3) (5,10) (6,0) (8,19) (11,7) (14,9) (15,4) (16,18) (17,13)

slot 6 (0,19) (3,16) (6,5) (7,1) (9,4) (10,8) (11,2) (13,15) (17,14) (18,12)
slot 7 (1,10) (4,5) (7,8) (12,6) (13,11) (14,0) (15,9) (16,17) (18,3) (19,2)
slot 8 (0,15) (2,6) (3,13) (4,8) (5,18) (9,19) (10,14) (11,16) (12,7) (17,1)
slot 9 (0,7) (2,17) (3,15) (6,18) (8,5) (9,16) (10,4) (13,12) (14,1) (19,11)
slot 10 (1,3) (4,0) (5,14) (7,19) (8,11) (12,9) (13,2) (15,18) (16,6) (17,10)
slot 11 (0,17) (1,8) (3,6) (5,12) (9,13) (11,10) (14,7) (16,15) (18,2) (19,4)
slot 12 (2,9) (4,3) (6,1) (8,0) (10,7) (11,18) (12,17) (13,14) (15,5) (19,16)
slot 13 (0,12) (1,16) (2,10) (3,7) (4,13) (5,19) (6,15) (8,17) (9,11) (18,14)
slot 14 (1,2) (3,12) (5,13) (7,4) (9,8) (10,6) (14,11) (15,17) (16,0) (19,18)
slot 15 (0,3) (2,5) (4,1) (6,17) (7,13) (8,14) (10,19) (11,15) (12,16) (18,9)
slot 16 (1,18) (2,14) (3,19) (9,6) (11,0) (12,10) (13,8) (15,7) (16,4) (17,5)
slot 17 (0,1) (4,2) (5,16) (6,11) (7,9) (8,3) (13,19) (14,12) (15,10) (17,18)
slot 18 (2,16) (3,17) (6,14) (7,5) (9,0) (10,13) (11,1) (12,4) (18,8) (19,15)
slot 19 (0,2) (4,14) (9,12) (10,5) (13,3) (15,1) (16,11) (17,8) (18,7) (19,6)
slot 20 (1,9) (3,18) (5,17) (7,6) (8,10) (11,19) (12,0) (13,4) (14,2) (15,16)
slot 21 (1,14) (2,18) (4,15) (6,13) (7,12) (8,9) (10,0) (11,3) (16,5) (17,19)
slot 22 (0,8) (2,4) (5,1) (6,16) (9,15) (12,3) (13,7) (14,17) (18,11) (19,10)
slot 23 (1,0) (3,9) (4,7) (8,16) (10,2) (11,14) (13,17) (15,6) (18,5) (19,12)
slot 24 (2,19) (4,10) (5,15) (6,8) (7,3) (11,9) (12,18) (14,13) (16,1) (17,0)
slot 25 (1,17) (3,14) (5,11) (6,4) (8,2) (9,18) (10,15) (13,0) (16,12) (19,7)
slot 26 (0,16) (3,2) (7,11) (8,13) (9,10) (12,15) (14,6) (17,4) (18,1) (19,5)
slot 27 (0,4) (1,7) (2,13) (5,8) (10,3) (11,12) (15,14) (16,9) (17,6) (18,19)
slot 28 (2,1) (3,0) (4,16) (6,12) (7,17) (9,5) (11,8) (13,10) (14,18) (15,19)
slot 29 (0,9) (1,19) (3,5) (6,2) (7,15) (8,4) (10,11) (12,14) (13,18) (17,16)
slot 30 (0,11) (1,6) (4,12) (5,2) (9,7) (14,10) (15,13) (16,3) (18,17) (19,8)
slot 31 (4,9) (7,14) (8,1) (11,13) (12,5) (15,0) (16,10) (17,2) (18,6) (19,3)
slot 32 (0,5) (1,11) (2,7) (3,4) (6,9) (8,12) (10,18) (13,16) (14,19) (17,15)
slot 33 (2,11) (3,1) (4,18) (5,6) (7,0) (9,14) (10,17) (12,13) (15,8) (16,19)
slot 34 (1,13) (5,4) (6,10) (8,7) (11,17) (12,2) (14,16) (15,3) (18,0) (19,9)
slot 35 (2,15) (4,11) (6,3) (8,18) (10,1) (13,9) (14,5) (16,7) (17,12) (19,0)
slot 36 (0,14) (3,8) (4,19) (7,10) (9,17) (11,6) (12,1) (13,5) (16,2) (18,15)
slot 37 (0,6) (1,4) (5,7) (9,2) (10,12) (14,8) (15,11) (17,3) (18,16) (19,13)

Πρόβλημα Middle 13

Η λύση που βρήκαμε είναι (0, 4715) με ανάλυση κόστους:

CA1: 15

CA2: 480

CA4: 370

SE1: 3850

και ωρολόγιο πρόγραμμα:

slot 0 (0,2) (1,19) (3,15) (4,12) (5,13) (6,10) (7,11) (8,16) (9,14) (17,18)
slot 1 (1,6) (2,17) (7,14) (9,10) (11,4) (12,5) (13,0) (15,16) (18,8) (19,3)
slot 2 (0,13) (4,9) (10,11) (12,3) (14,6) (15,8) (16,1) (17,7) (18,5) (19,2)
slot 3 (3,7) (4,1) (5,11) (6,9) (8,15) (10,14) (12,2) (13,19) (17,16) (18,0)
slot 4 (1,14) (2,10) (5,4) (6,3) (7,13) (9,8) (11,17) (12,15) (16,0) (19,18)
slot 5 (0,17) (1,12) (5,6) (7,19) (8,3) (10,13) (11,9) (14,16) (15,4) (18,2)
slot 6 (0,1) (3,6) (4,10) (11,8) (13,12) (15,7) (16,2) (17,5) (18,9) (19,14)
slot 7 (2,1) (3,17) (5,15) (6,11) (8,4) (9,18) (10,16) (12,7) (13,14) (19,0)
slot 8 (1,15) (3,2) (5,19) (6,12) (7,4) (8,18) (9,17) (10,0) (14,13) (16,11)
slot 9 (0,8) (1,10) (4,2) (11,3) (12,14) (13,9) (15,6) (16,5) (17,19) (18,7)
slot 10 (1,17) (2,13) (3,16) (7,10) (8,12) (9,5) (14,11) (15,0) (18,4) (19,6)
slot 11 (4,16) (5,18) (6,2) (7,1) (8,14) (9,11) (10,3) (12,0) (13,17) (19,15)
slot 12 (0,6) (1,2) (3,13) (7,5) (9,12) (11,19) (14,8) (16,15) (17,4) (18,10)
slot 13 (1,8) (2,14) (4,11) (5,3) (10,6) (12,19) (13,18) (15,9) (16,7) (17,0)
slot 14 (0,14) (6,7) (8,5) (9,3) (10,2) (11,13) (12,4) (17,15) (18,1) (19,16)
slot 15 (3,1) (4,17) (5,9) (7,6) (11,0) (12,8) (13,10) (14,18) (15,2) (16,19)
slot 16 (0,18) (1,16) (2,7) (6,15) (8,11) (12,10) (13,4) (14,3) (17,9) (19,5)
slot 17 (0,12) (1,11) (4,15) (9,6) (10,8) (13,2) (14,5) (16,3) (18,17) (19,7)
slot 18 (0,4) (3,10) (5,12) (7,18) (8,6) (11,16) (14,9) (15,13) (17,2) (19,1)
slot 19 (2,5) (3,11) (4,13) (6,1) (8,0) (10,7) (12,17) (15,14) (18,16) (19,9)
slot 20 (1,3) (2,18) (4,6) (5,0) (9,7) (11,10) (13,15) (14,19) (16,8) (17,12)
slot 21 (0,16) (2,9) (3,4) (5,1) (7,15) (8,13) (10,19) (12,11) (17,6) (18,14)
slot 22 (0,9) (1,18) (2,3) (5,17) (6,8) (11,7) (14,4) (15,10) (16,12) (19,13)
slot 23 (3,19) (4,5) (7,8) (9,2) (10,17) (11,15) (12,18) (13,1) (14,0) (16,6)
slot 24 (0,7) (5,16) (6,4) (8,2) (9,15) (13,11) (14,12) (17,1) (18,3) (19,10)
slot 25 (1,0) (2,4) (3,12) (6,5) (7,17) (8,9) (11,14) (15,19) (16,10) (18,13)
slot 26 (0,19) (2,8) (4,14) (7,9) (10,5) (12,6) (13,16) (15,1) (17,3) (18,11)
slot 27 (3,0) (5,2) (6,16) (8,7) (9,13) (10,15) (14,1) (17,11) (18,12) (19,4)
slot 28 (0,10) (1,7) (2,19) (3,9) (6,14) (8,17) (11,5) (12,13) (15,18) (16,4)

slot 29 (2,0) (3,5) (4,19) (7,16) (9,1) (10,12) (11,18) (13,6) (14,15) (17,8)
slot 30 (0,11) (4,8) (5,14) (7,2) (10,1) (12,9) (15,3) (16,13) (18,6) (19,17)
slot 31 (0,15) (1,5) (4,18) (6,13) (7,3) (8,10) (9,16) (11,2) (17,14) (19,12)
slot 32 (2,16) (3,18) (4,0) (5,7) (6,19) (10,9) (12,1) (13,8) (14,17) (15,11)
slot 33 (2,6) (3,14) (7,12) (9,0) (10,4) (11,1) (13,5) (16,17) (18,15) (19,8)
slot 34 (0,5) (1,9) (2,15) (4,7) (6,18) (8,19) (11,12) (13,3) (16,14) (17,10)
slot 35 (1,13) (3,8) (5,10) (7,0) (9,4) (11,6) (12,16) (14,2) (15,17) (18,19)
slot 36 (2,12) (4,3) (6,0) (8,1) (10,18) (14,7) (15,5) (16,9) (17,13) (19,11)
slot 37 (0,3) (1,4) (2,11) (5,8) (6,17) (9,19) (13,7) (14,10) (15,12) (16,18)

Πρόβλημα Middle 14

Η λύση που βρήκαμε είναι (0, 2889) με ανάλυση κόστους:

CA1: 10
CA2: 680
CA3: 405
CA4: 1775
GA1: 9
FA2: 10

και ωρολόγιο πρόγραμμα:

slot 0 (0,1) (3,19) (4,2) (6,10) (7,9) (11,5) (13,12) (15,17) (16,14) (18,8)
slot 1 (1,4) (2,6) (5,7) (8,3) (9,0) (10,16) (12,13) (14,15) (17,11) (19,18)
slot 2 (0,17) (3,14) (4,12) (8,19) (9,2) (11,7) (13,10) (15,1) (16,6) (18,5)
slot 3 (1,3) (2,15) (5,11) (6,16) (7,18) (10,0) (12,9) (14,13) (17,4) (19,8)
slot 4 (0,14) (3,5) (4,6) (8,7) (9,12) (13,1) (15,2) (16,17) (18,11) (19,10)
slot 5 (1,19) (2,9) (4,16) (5,12) (6,0) (7,15) (10,13) (11,3) (14,18) (17,8)
slot 6 (0,2) (1,10) (9,6) (11,8) (12,18) (13,15) (14,5) (16,3) (17,7) (19,4)
slot 7 (2,1) (3,11) (4,0) (5,16) (6,12) (7,17) (8,14) (10,9) (15,19) (18,13)
slot 8 (0,3) (1,5) (8,11) (9,15) (12,2) (13,6) (14,4) (16,10) (18,7) (19,17)
slot 9 (2,8) (3,4) (5,13) (6,19) (7,1) (10,18) (11,14) (15,12) (16,9) (17,0)
slot 10 (0,12) (1,11) (3,2) (4,15) (7,6) (8,5) (13,9) (14,17) (18,10) (19,16)
slot 11 (2,19) (5,3) (6,14) (8,18) (9,13) (11,10) (12,4) (15,0) (16,7) (17,1)
slot 12 (0,18) (1,15) (3,9) (4,8) (5,2) (7,16) (10,11) (13,17) (14,12) (19,6)
slot 13 (2,14) (6,15) (8,4) (9,7) (10,5) (11,13) (12,19) (16,0) (17,3) (18,1)
slot 14 (0,11) (1,6) (3,18) (4,9) (7,10) (12,17) (13,2) (14,8) (15,16) (19,5)
slot 15 (2,16) (5,19) (6,13) (7,4) (8,0) (10,1) (11,12) (14,9) (17,15) (18,3)
slot 16 (0,10) (1,2) (3,6) (4,17) (9,8) (12,14) (13,11) (15,5) (16,18) (19,7)

slot 17 (0,13) (2,7) (5,1) (6,4) (8,12) (10,19) (11,9) (14,3) (17,16) (18,15)
slot 18 (3,17) (4,5) (7,0) (9,10) (12,6) (13,8) (15,14) (16,11) (18,2) (19,1)
slot 19 (1,18) (5,0) (6,2) (8,13) (10,12) (11,15) (14,7) (16,4) (17,9) (19,3)
slot 20 (0,16) (2,10) (3,8) (7,14) (9,1) (12,11) (13,19) (15,6) (17,5) (18,4)
slot 21 (1,0) (4,7) (5,17) (6,3) (8,9) (11,18) (14,10) (15,13) (16,2) (19,12)
slot 22 (0,4) (2,11) (3,1) (7,8) (9,5) (10,6) (12,15) (13,14) (16,19) (17,18)
slot 23 (1,9) (3,12) (5,4) (6,7) (8,16) (11,17) (14,2) (15,10) (18,0) (19,13)
slot 24 (0,5) (4,1) (6,8) (7,11) (9,14) (10,2) (12,3) (13,18) (16,15) (17,19)
slot 25 (1,13) (2,0) (3,7) (4,11) (5,18) (8,6) (9,17) (12,10) (14,16) (19,15)
slot 26 (0,6) (2,4) (5,9) (7,12) (10,3) (11,1) (15,8) (16,13) (17,14) (18,19)
slot 27 (1,14) (3,0) (4,18) (6,11) (8,15) (9,16) (10,17) (12,7) (13,5) (19,2)
slot 28 (0,8) (2,13) (5,10) (7,3) (11,4) (14,19) (15,9) (16,1) (17,6) (18,12)
slot 29 (2,5) (4,3) (6,1) (8,17) (9,18) (10,14) (12,16) (13,0) (15,7) (19,11)
slot 30 (0,9) (1,8) (3,15) (4,10) (5,6) (7,2) (11,19) (16,12) (17,13) (18,14)
slot 31 (2,17) (6,18) (8,1) (9,3) (10,7) (12,5) (13,16) (14,11) (15,4) (19,0)
slot 32 (0,15) (1,17) (3,10) (4,19) (5,14) (6,9) (7,13) (11,2) (12,8) (18,16)
slot 33 (0,19) (2,3) (9,4) (10,8) (13,7) (14,1) (15,11) (16,5) (17,12) (18,6)
slot 34 (1,7) (2,18) (3,13) (4,14) (5,15) (6,17) (8,10) (11,16) (12,0) (19,9)
slot 35 (1,12) (7,5) (9,19) (10,4) (11,0) (13,3) (14,6) (15,18) (16,8) (17,2)
slot 36 (0,7) (3,16) (4,13) (6,5) (8,2) (9,11) (10,15) (12,1) (18,17) (19,14)
slot 37 (1,16) (2,12) (5,8) (7,19) (11,6) (13,4) (14,0) (15,3) (17,10) (18,9)

Πρόβλημα Middle 15

Η λύση που βρήκαμε είναι (0, 6290) με ανάλυση κόστους:

CA3: 430
GA1: 25
BR1: 5
BR2: 2180
SE1: 3650

και ωρολόγιο πρόγραμμα:

slot 0 (0,3) (4,1) (7,17) (9,5) (10,6) (11,16) (12,8) (15,2) (18,13) (19,14)
slot 1 (1,18) (5,4) (6,2) (7,11) (8,19) (9,12) (13,15) (14,0) (16,3) (17,10)
slot 2 (0,9) (2,8) (4,16) (5,17) (7,15) (10,13) (11,3) (12,6) (14,1) (19,18)
slot 3 (1,11) (2,14) (3,10) (5,16) (7,8) (9,6) (12,4) (13,0) (15,19) (17,18)
slot 4 (0,17) (1,5) (3,9) (4,19) (6,8) (10,7) (11,2) (12,18) (14,16) (15,13)
slot 5 (1,3) (4,2) (5,12) (7,6) (8,9) (10,0) (13,18) (16,15) (17,14) (19,11)

slot 6 (2,1) (3,0) (8,4) (9,17) (11,13) (12,19) (14,5) (15,7) (16,10) (18,6)
slot 7 (1,7) (2,19) (4,9) (5,18) (8,6) (10,14) (12,3) (13,11) (15,17) (16,0)
slot 8 (3,4) (6,13) (7,12) (8,5) (9,15) (10,1) (14,2) (17,11) (18,16) (19,0)
slot 9 (0,12) (1,8) (2,3) (5,9) (11,6) (14,13) (15,4) (16,17) (18,7) (19,10)
slot 10 (1,0) (4,11) (5,13) (6,16) (7,3) (8,17) (10,15) (12,2) (14,19) (18,9)
slot 11 (0,18) (2,13) (3,8) (6,15) (7,5) (9,14) (10,12) (16,11) (17,1) (19,4)
slot 12 (0,6) (1,16) (3,13) (5,19) (9,7) (12,11) (14,8) (15,10) (17,2) (18,4)
slot 13 (3,19) (4,14) (6,17) (8,12) (10,9) (11,0) (13,7) (15,1) (16,2) (18,5)
slot 14 (1,12) (2,6) (4,15) (5,0) (7,10) (9,3) (11,8) (13,17) (14,18) (19,16)
slot 15 (1,2) (3,12) (4,13) (6,5) (8,7) (9,19) (11,10) (15,18) (16,14) (17,0)
slot 16 (0,11) (1,4) (2,15) (5,10) (6,12) (7,19) (9,16) (13,3) (14,17) (18,8)
slot 17 (3,1) (4,5) (7,0) (8,11) (10,18) (12,16) (13,2) (15,14) (17,6) (19,9)
slot 18 (0,5) (1,14) (3,16) (4,17) (6,10) (7,13) (8,15) (9,2) (11,12) (18,19)
slot 19 (2,0) (3,11) (5,14) (6,7) (8,16) (12,10) (13,4) (15,9) (18,1) (19,17)
slot 20 (0,1) (2,18) (4,8) (7,9) (10,19) (13,12) (14,11) (15,5) (16,6) (17,3)
slot 21 (1,13) (3,18) (5,6) (9,8) (10,2) (11,7) (14,4) (15,0) (17,16) (19,12)
slot 22 (0,4) (1,19) (6,14) (7,2) (11,17) (12,9) (13,8) (15,3) (16,5) (18,10)
slot 23 (2,16) (4,7) (5,15) (6,0) (9,1) (10,3) (12,14) (17,8) (18,11) (19,13)
slot 24 (0,7) (1,17) (2,12) (4,10) (5,8) (6,11) (13,9) (14,15) (16,18) (19,3)
slot 25 (3,2) (7,18) (9,0) (10,8) (11,14) (12,1) (13,5) (15,6) (16,19) (17,4)
slot 26 (0,2) (1,10) (4,3) (5,11) (8,13) (14,6) (16,7) (17,9) (18,12) (19,15)
slot 27 (2,5) (3,15) (7,14) (8,0) (9,18) (11,1) (12,17) (13,10) (16,4) (19,6)
slot 28 (0,13) (5,1) (6,19) (8,3) (9,4) (10,11) (14,12) (15,16) (17,7) (18,2)
slot 29 (1,6) (3,14) (7,4) (9,13) (10,17) (11,5) (12,15) (16,8) (18,0) (19,2)
slot 30 (0,10) (2,7) (4,12) (5,3) (6,18) (8,1) (13,16) (14,9) (15,11) (17,19)
slot 31 (0,15) (2,10) (3,5) (6,9) (11,4) (12,7) (16,1) (17,13) (18,14) (19,8)
slot 32 (1,15) (3,17) (4,6) (5,2) (7,16) (8,18) (9,11) (12,0) (13,19) (14,10)
slot 33 (0,8) (2,9) (6,4) (10,16) (11,18) (13,1) (14,3) (15,12) (17,5) (19,7)
slot 34 (0,16) (2,4) (3,6) (7,1) (8,10) (11,9) (13,14) (17,12) (18,15) (19,5)
slot 35 (0,19) (2,17) (5,7) (6,1) (8,14) (10,4) (11,15) (12,13) (16,9) (18,3)
slot 36 (2,11) (4,0) (6,3) (9,10) (12,5) (14,7) (15,8) (16,13) (18,17) (19,1)
slot 37 (0,14) (1,9) (3,7) (4,18) (8,2) (10,5) (11,19) (13,6) (16,12) (17,15)

Πρόβλημα Late 1

Βρήκαμε την λύση (0, 3115) που έχει ανάλυση κόστους

CA1: 23

CA2: 665

CA3: 95

CA4: 2240

GA1: 2

FA2: 90

και ωρολόγιο πρόγραμμα:

slot 0 (1,3) (4,12) (5,15) (7,2) (9,0) (11,6) (13,8) (14,10)
slot 1 (1,12) (2,0) (3,4) (5,13) (6,14) (8,10) (11,7) (15,9)
slot 2 (0,15) (2,1) (3,11) (7,4) (9,5) (10,6) (12,13) (14,8)
slot 3 (1,10) (4,14) (5,7) (6,0) (8,3) (9,2) (13,12) (15,11)
slot 4 (1,11) (2,9) (4,8) (6,3) (10,7) (12,14) (13,0) (15,5)
slot 5 (0,7) (3,1) (5,2) (8,6) (9,15) (11,13) (12,10) (14,4)
slot 6 (0,11) (2,12) (4,10) (5,6) (7,15) (8,9) (13,3) (14,1)
slot 7 (1,6) (3,13) (4,7) (9,8) (10,2) (11,5) (12,0) (15,14)
slot 8 (0,14) (2,4) (3,5) (7,12) (10,1) (11,8) (13,9) (15,6)
slot 9 (0,3) (4,1) (5,9) (6,10) (7,8) (12,2) (13,15) (14,11)
slot 10 (1,0) (2,8) (4,11) (9,6) (10,5) (12,7) (14,3) (15,13)
slot 11 (1,13) (3,12) (6,5) (8,7) (9,14) (10,4) (11,2) (15,0)
slot 12 (0,6) (2,15) (4,13) (7,3) (8,1) (11,10) (12,5) (14,9)
slot 13 (1,15) (2,11) (5,3) (6,12) (9,4) (10,8) (13,7) (14,0)
slot 14 (0,13) (3,14) (4,5) (6,7) (8,2) (10,12) (11,9) (15,1)
slot 15 (0,4) (1,8) (2,14) (5,10) (7,9) (11,3) (12,15) (13,6)
slot 16 (3,8) (4,0) (6,15) (7,11) (10,9) (12,1) (13,2) (14,5)
slot 17 (0,12) (1,7) (2,10) (3,6) (5,4) (8,13) (9,11) (14,15)
slot 18 (4,2) (6,1) (7,14) (9,10) (11,0) (12,3) (13,5) (15,8)
slot 19 (0,9) (3,7) (4,6) (5,11) (8,12) (10,15) (13,1) (14,2)
slot 20 (0,5) (2,13) (3,10) (6,9) (11,1) (12,8) (14,7) (15,4)
slot 21 (5,1) (6,2) (7,0) (8,14) (9,13) (10,11) (12,4) (15,3)
slot 22 (0,8) (2,6) (4,15) (5,14) (7,1) (9,12) (10,3) (13,11)
slot 23 (0,10) (1,9) (2,3) (6,4) (8,5) (11,12) (14,13) (15,7)
slot 24 (1,14) (5,0) (6,8) (7,10) (9,3) (12,11) (13,4) (15,2)
slot 25 (0,1) (3,2) (4,9) (5,12) (7,6) (8,15) (10,13) (11,14)
slot 26 (1,4) (2,5) (3,0) (6,13) (8,11) (9,7) (14,12) (15,10)

slot 27 (0,2) (4,3) (5,8) (6,11) (7,13) (9,1) (10,14) (15,12)
slot 28 (1,5) (2,7) (3,9) (8,4) (10,0) (11,15) (12,6) (13,14)
slot 29 (1,2) (3,15) (7,5) (8,0) (11,4) (12,9) (13,10) (14,6)

Πρόβλημα Late 2

Βρήκαμε την λύση (0, 6434) που έχει ανάλυση κόστους

CA2: 2370
CA3: 1075
CA4: 2985
GA1: 4

και ωρολόγιο πρόγραμμα:

slot 0 (3,9) (5,1) (6,2) (10,7) (11,8) (12,15) (13,0) (14,4)
slot 1 (0,13) (1,6) (2,10) (4,14) (7,3) (9,8) (12,5) (15,11)
slot 2 (0,4) (3,11) (5,2) (6,7) (8,15) (10,12) (13,9) (14,1)
slot 3 (1,5) (3,10) (4,9) (7,0) (11,12) (13,6) (14,8) (15,2)
slot 4 (0,7) (1,4) (2,13) (5,11) (6,15) (8,14) (9,10) (12,3)
slot 5 (3,2) (4,12) (7,9) (10,5) (11,6) (13,8) (14,0) (15,1)
slot 6 (0,11) (1,3) (2,15) (5,14) (8,4) (9,13) (10,6) (12,7)
slot 7 (4,7) (6,10) (8,3) (9,0) (11,1) (13,2) (14,12) (15,5)
slot 8 (0,12) (1,9) (2,14) (3,8) (4,13) (5,10) (7,6) (11,15)
slot 9 (5,0) (6,1) (8,2) (9,3) (10,11) (12,4) (13,14) (15,7)
slot 10 (1,10) (2,5) (3,13) (4,6) (7,12) (11,0) (14,9) (15,8)
slot 11 (0,15) (1,8) (2,11) (5,7) (6,14) (9,12) (10,4) (13,3)
slot 12 (3,6) (4,1) (7,10) (8,0) (11,5) (12,2) (14,13) (15,9)
slot 13 (0,10) (1,11) (2,8) (4,15) (5,12) (6,13) (9,7) (14,3)
slot 14 (6,3) (7,2) (8,11) (9,5) (10,0) (12,14) (13,1) (15,4)
slot 15 (0,6) (2,4) (3,1) (5,8) (11,10) (12,9) (14,7) (15,13)
slot 16 (1,14) (3,0) (4,11) (6,12) (7,15) (8,9) (10,2) (13,5)
slot 17 (0,14) (2,3) (5,4) (7,1) (9,6) (11,13) (12,8) (15,10)
slot 18 (1,2) (3,15) (4,10) (6,0) (8,7) (9,11) (13,12) (14,5)
slot 19 (3,14) (6,9) (7,4) (8,5) (10,1) (11,2) (12,13) (15,0)
slot 20 (0,3) (1,12) (2,7) (4,8) (5,6) (9,15) (10,14) (13,11)
slot 21 (0,1) (3,4) (7,5) (8,13) (11,9) (12,10) (14,2) (15,6)
slot 22 (1,7) (2,0) (5,3) (6,11) (8,12) (9,14) (10,15) (13,4)
slot 23 (0,2) (4,5) (7,8) (9,1) (10,13) (12,6) (14,11) (15,3)
slot 24 (1,0) (2,9) (3,12) (5,15) (6,4) (8,10) (11,14) (13,7)

slot 25 (0,5) (1,15) (4,3) (7,13) (9,2) (10,8) (12,11) (14,6)
slot 26 (0,9) (2,12) (3,7) (6,5) (8,1) (11,4) (13,10) (15,14)
slot 27 (1,13) (4,2) (5,9) (6,8) (7,11) (10,3) (12,0) (14,15)
slot 28 (0,8) (2,6) (3,5) (9,4) (11,7) (12,1) (13,15) (14,10)
slot 29 (2,1) (4,0) (5,13) (7,14) (8,6) (10,9) (11,3) (15,12)

Πρόβλημα Late 3

Βρήκαμε την λύση (0, 2833) που έχει ανάλυση κόστους

CA2: 985
CA3: 280
CA4: 455
GA1: 3
BR2: 1000
SE1: 110

και ωρολόγιο πρόγραμμα:

slot 0 (2,1) (4,5) (6,8) (7,3) (9,13) (12,10) (14,11) (15,0)
slot 1 (0,6) (1,12) (3,8) (9,4) (10,7) (11,13) (14,2) (15,5)
slot 2 (0,9) (2,10) (4,3) (6,14) (8,1) (11,7) (12,5) (13,15)
slot 3 (1,15) (3,0) (4,10) (5,6) (7,12) (8,14) (9,11) (13,2)
slot 4 (1,0) (2,8) (5,13) (6,9) (11,3) (12,4) (14,7) (15,10)
slot 5 (0,2) (3,5) (4,7) (6,1) (8,9) (10,11) (13,14) (15,12)
slot 6 (1,13) (2,6) (5,7) (9,15) (10,0) (11,4) (12,8) (14,3)
slot 7 (1,10) (2,11) (3,9) (5,14) (6,12) (7,15) (8,4) (13,0)
slot 8 (0,12) (7,2) (9,5) (10,3) (11,6) (13,8) (14,1) (15,4)
slot 9 (1,7) (2,9) (3,15) (4,6) (5,11) (8,0) (10,13) (12,14)
slot 10 (0,4) (2,12) (7,6) (8,5) (9,1) (13,3) (14,10) (15,11)
slot 11 (1,4) (5,2) (6,3) (7,9) (10,8) (11,0) (12,13) (14,15)
slot 12 (0,14) (3,2) (4,13) (5,1) (8,7) (9,10) (11,12) (15,6)
slot 13 (1,11) (2,4) (6,13) (7,0) (8,15) (10,5) (12,3) (14,9)
slot 14 (0,5) (3,1) (4,14) (6,10) (11,8) (12,9) (13,7) (15,2)
slot 15 (1,2) (4,12) (5,3) (7,14) (8,6) (9,0) (10,15) (13,11)
slot 16 (0,1) (2,13) (3,4) (5,15) (9,8) (11,10) (12,7) (14,6)
slot 17 (2,0) (3,7) (4,11) (6,5) (8,12) (10,14) (13,9) (15,1)
slot 18 (0,15) (1,8) (6,11) (7,4) (9,3) (10,2) (13,5) (14,12)
slot 19 (1,6) (2,14) (3,11) (4,8) (5,9) (7,10) (12,0) (15,13)
slot 20 (0,3) (6,4) (8,2) (10,12) (11,9) (13,1) (14,5) (15,7)

slot 21 (0,10) (1,14) (3,13) (4,15) (5,8) (7,11) (9,2) (12,6)
slot 22 (1,9) (6,2) (7,5) (8,3) (10,4) (11,15) (13,12) (14,0)
slot 23 (0,8) (2,15) (3,6) (5,4) (9,7) (10,1) (12,11) (14,13)
slot 24 (3,14) (4,0) (5,10) (6,7) (8,13) (11,2) (12,1) (15,9)
slot 25 (0,11) (2,5) (7,1) (9,12) (10,6) (13,4) (14,8) (15,3)
slot 26 (0,13) (1,3) (2,7) (4,9) (5,12) (6,15) (8,10) (11,14)
slot 27 (3,10) (5,0) (7,13) (9,6) (11,1) (12,2) (14,4) (15,8)
slot 28 (0,7) (1,5) (3,12) (4,2) (8,11) (10,9) (13,6) (15,14)
slot 29 (2,3) (4,1) (6,0) (7,8) (9,14) (11,5) (12,15) (13,10)

Πρόβλημα Late 4

Η λύση που βρήκαμε είναι (0, 0) με ωρολόγιο πρόγραμμα:

slot 0 (0,14) (1,8) (3,11) (4,7) (5,16) (6,12) (10,17) (13,9) (15,2)
slot 1 (0,5) (1,17) (3,13) (6,9) (7,16) (10,15) (11,8) (12,2) (14,4)
slot 2 (0,8) (1,3) (5,6) (10,2) (11,14) (13,7) (15,4) (16,9) (17,12)
slot 3 (1,14) (4,3) (5,10) (6,16) (7,2) (8,13) (9,12) (11,15) (17,0)
slot 4 (0,15) (5,3) (6,17) (8,2) (9,7) (10,4) (11,12) (13,14) (16,1)
slot 5 (1,10) (4,17) (6,0) (7,14) (9,2) (12,8) (13,11) (15,5) (16,3)
slot 6 (1,12) (2,4) (3,7) (5,17) (9,15) (11,0) (13,10) (14,6) (16,8)
slot 7 (1,13) (2,11) (3,9) (4,6) (7,0) (8,17) (14,5) (15,12) (16,10)
slot 8 (3,0) (7,5) (8,14) (9,1) (10,12) (11,16) (13,4) (15,6) (17,2)
slot 9 (2,6) (3,8) (4,0) (7,1) (11,9) (13,5) (14,10) (16,12) (17,15)
slot 10 (1,6) (2,5) (4,11) (7,8) (9,0) (10,3) (12,13) (14,17) (15,16)
slot 11 (0,12) (5,11) (6,3) (8,4) (10,7) (14,9) (15,1) (16,2) (17,13)
slot 12 (0,2) (3,14) (5,8) (9,17) (10,6) (11,1) (12,7) (15,13) (16,4)
slot 13 (0,1) (2,13) (4,5) (7,6) (9,8) (10,11) (14,12) (15,3) (17,16)
slot 14 (3,12) (4,9) (5,1) (7,17) (10,0) (11,6) (14,2) (15,8) (16,13)
slot 15 (0,13) (2,3) (4,1) (6,8) (7,15) (10,9) (11,17) (12,5) (14,16)
slot 16 (0,16) (2,1) (3,17) (4,12) (7,11) (9,5) (10,8) (13,6) (15,14)
slot 17 (4,2) (7,3) (8,0) (9,13) (12,6) (14,1) (15,11) (16,5) (17,10)
slot 18 (0,7) (1,16) (2,12) (4,14) (6,5) (10,13) (11,3) (15,9) (17,8)
slot 19 (2,15) (4,13) (5,0) (8,11) (9,3) (10,16) (12,1) (14,7) (17,6)
slot 20 (0,11) (2,17) (3,4) (5,7) (6,14) (8,12) (9,16) (13,1) (15,10)
slot 21 (0,3) (1,7) (2,9) (4,10) (6,15) (11,13) (12,16) (14,8) (17,5)
slot 22 (0,9) (1,15) (2,10) (3,6) (5,14) (7,4) (12,17) (13,8) (16,11)
slot 23 (5,13) (6,2) (8,3) (9,11) (10,1) (12,15) (14,0) (16,7) (17,4)

slot 24 (0,17) (1,9) (2,8) (5,4) (6,10) (7,13) (12,11) (14,3) (16,15)
slot 25 (0,4) (3,5) (7,10) (8,1) (11,2) (12,14) (13,15) (16,6) (17,9)
slot 26 (2,0) (4,8) (6,1) (9,14) (10,5) (12,3) (13,16) (15,7) (17,11)
slot 27 (1,11) (3,15) (5,12) (6,4) (8,9) (10,14) (13,2) (16,0) (17,7)
slot 28 (1,0) (2,16) (3,10) (6,11) (7,12) (8,5) (9,4) (14,13) (15,17)
slot 29 (0,6) (3,1) (4,15) (5,2) (7,9) (8,16) (11,10) (13,12) (17,14)
slot 30 (1,4) (2,7) (8,15) (9,6) (11,5) (12,10) (13,0) (16,14) (17,3)
slot 31 (0,10) (1,5) (3,2) (4,16) (8,6) (11,7) (12,9) (13,17) (14,15)
slot 32 (2,14) (3,16) (5,15) (6,13) (8,7) (9,10) (11,4) (12,0) (17,1)
slot 33 (1,2) (5,9) (6,7) (8,10) (12,4) (13,3) (14,11) (15,0) (16,17)

Η λύση (0, 0) που βρήκαμε μηδενίζει το κόστος των ασθενών περιορισμών, δηλαδή δεν υπάρχει καμία παραβίαση των ασθενών περιορισμών και συνεπώς, ικανοποιούνται όλοι. Έτσι, η λύση αυτή αποτελεί ολικό βέλτιστο του προβλήματος Late 4.

Όμως, αν και η παραπάνω λύση έχει την ίδια τιμή αντικειμενικής συνάρτησης με την βέλτιστη δημοσιευμένη από τον διαγωνισμό ITC2021 λύση, το δικό μας ωρολόγιο πρόγραμμα είναι **διαφορετικό**. Μάλιστα, η απόσταση d της δικής μας λύσης από την λύση του διαγωνισμού (σχέση 55) ισούται με $d = 0.9118$ που σημαίνει ότι η λύση που βρήκαμε είναι μια εντελώς διαφορετική λύση από εκείνη του διαγωνισμού.

Αυτό σημαίνει ότι το πρόβλημα Late 4 έχει (τουλάχιστον) δύο διαφορετικά ολικά βέλτιστα.

Πρόβλημα Late 5

Το αποτέλεσμα που βρήκαμε (29, 2648), το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους

Ισχυροί περιορισμοί
176 = 174 ικανοποιούνται + 2 παραβιάζονται
CA4: 1
BR2: 28
Ασθενείς περιορισμοί
CA1: 16
CA2: 1800
CA4: 750
GA1: 2

FA2: 80

και ωρολόγιο πρόγραμμα:

slot 0 (1,8) (2,12) (3,11) (4,16) (5,14) (6,17) (7,0) (10,9) (15,13)
slot 1 (5,6) (8,15) (9,7) (11,2) (12,1) (13,3) (14,4) (16,10) (17,0)
slot 2 (0,11) (1,7) (3,15) (4,2) (6,9) (10,14) (12,8) (13,5) (17,16)
slot 3 (2,3) (4,10) (5,0) (7,11) (8,17) (9,13) (14,6) (15,1) (16,12)
slot 4 (0,10) (2,1) (3,14) (6,7) (9,4) (11,8) (12,17) (13,16) (15,5)
slot 5 (0,15) (1,5) (4,3) (7,2) (10,6) (11,14) (12,13) (16,8) (17,9)
slot 6 (1,0) (2,16) (3,10) (4,11) (5,12) (8,9) (13,7) (14,17) (15,6)
slot 7 (0,16) (3,17) (6,2) (7,12) (8,4) (9,15) (10,1) (11,5) (13,14)
slot 8 (1,6) (2,8) (7,4) (10,5) (11,9) (12,0) (14,15) (16,3) (17,13)
slot 9 (0,2) (3,9) (4,6) (5,17) (8,13) (12,11) (14,7) (15,10) (16,1)
slot 10 (1,4) (2,5) (3,8) (6,11) (7,17) (9,14) (10,12) (13,0) (15,16)
slot 11 (0,4) (2,9) (5,7) (8,10) (11,16) (12,3) (13,6) (14,1) (17,15)
slot 12 (0,14) (1,9) (3,5) (4,12) (6,8) (10,13) (15,11) (16,7) (17,2)
slot 13 (2,10) (5,4) (7,3) (8,14) (9,0) (11,17) (13,1) (15,12) (16,6)
slot 14 (0,8) (1,17) (4,15) (5,16) (6,3) (7,10) (11,13) (12,9) (14,2)
slot 15 (3,1) (6,0) (8,5) (9,16) (10,11) (13,2) (14,12) (15,7) (17,4)
slot 16 (0,3) (1,11) (2,15) (4,13) (5,9) (7,8) (12,6) (16,14) (17,10)
slot 17 (3,16) (4,7) (6,12) (8,0) (9,5) (10,2) (11,1) (13,17) (15,14)
slot 18 (0,6) (1,13) (2,7) (5,8) (10,4) (12,16) (14,3) (15,9) (17,11)
slot 19 (3,12) (4,14) (7,5) (8,1) (9,10) (11,0) (13,15) (16,2) (17,6)
slot 20 (0,17) (1,16) (5,13) (6,4) (8,2) (9,12) (10,7) (14,11) (15,3)
slot 21 (2,0) (4,8) (6,15) (7,1) (10,16) (11,3) (12,5) (13,9) (17,14)
slot 22 (1,12) (3,4) (5,10) (7,13) (8,6) (9,17) (14,0) (15,2) (16,11)
slot 23 (0,5) (2,6) (3,7) (4,9) (11,15) (12,10) (13,8) (14,16) (17,1)
slot 24 (2,17) (5,3) (6,1) (7,14) (8,12) (9,11) (10,0) (15,4) (16,13)
slot 25 (0,7) (1,10) (3,2) (4,5) (9,8) (11,6) (12,15) (14,13) (16,17)
slot 26 (2,11) (5,1) (6,14) (7,9) (8,16) (10,3) (13,4) (15,0) (17,12)
slot 27 (0,1) (3,13) (6,10) (9,2) (11,7) (12,4) (14,5) (16,15) (17,8)
slot 28 (1,3) (2,14) (4,0) (5,15) (7,6) (8,11) (10,17) (13,12) (16,9)
slot 29 (0,13) (1,2) (6,16) (9,3) (11,4) (12,7) (14,10) (15,8) (17,5)
slot 30 (3,6) (4,17) (5,2) (7,15) (9,1) (10,8) (12,14) (13,11) (16,0)
slot 31 (0,12) (1,15) (2,4) (6,13) (8,7) (11,10) (14,9) (16,5) (17,3)
slot 32 (3,0) (4,1) (5,11) (7,16) (9,6) (12,2) (13,10) (14,8) (15,17)
slot 33 (0,9) (1,14) (2,13) (6,5) (8,3) (10,15) (11,12) (16,4) (17,7)

Πρόβλημα Late 6

Βρήκαμε την λύση (0, 1648) που έχει ανάλυση κόστους

CA1: 23

CA2: 235

BR2: 1280

SE1: 110

και ωρολόγιο πρόγραμμα:

slot 0 (1,11) (3,6) (4,8) (5,14) (7,13) (9,12) (10,0) (16,15) (17,2)
slot 1 (2,7) (4,5) (8,17) (11,3) (12,1) (13,10) (14,6) (15,9) (16,0)
slot 2 (0,2) (1,14) (3,15) (5,8) (6,12) (7,10) (9,17) (11,16) (13,4)
slot 3 (2,3) (6,5) (8,7) (9,13) (10,11) (12,16) (14,0) (15,4) (17,1)
slot 4 (0,8) (1,10) (2,5) (3,12) (7,9) (11,4) (13,16) (15,14) (17,6)
slot 5 (0,17) (1,9) (4,12) (5,7) (6,15) (8,3) (10,14) (11,13) (16,2)
slot 6 (0,5) (2,13) (3,4) (7,1) (9,6) (12,8) (14,16) (15,10) (17,11)
slot 7 (3,1) (4,9) (5,13) (6,2) (8,14) (11,0) (12,7) (16,10) (17,15)
slot 8 (0,3) (1,15) (4,14) (7,6) (9,5) (10,2) (11,8) (13,12) (16,17)
slot 9 (3,13) (6,4) (8,1) (11,7) (12,0) (14,9) (15,2) (16,5) (17,10)
slot 10 (0,15) (1,13) (2,4) (5,17) (7,16) (8,6) (9,3) (10,12) (14,11)
slot 11 (0,9) (3,16) (4,7) (5,1) (6,11) (8,10) (14,2) (15,13) (17,12)
slot 12 (1,4) (2,8) (3,14) (7,0) (9,16) (10,6) (11,5) (12,15) (13,17)
slot 13 (2,9) (4,10) (5,3) (6,0) (8,13) (14,12) (15,11) (16,1) (17,7)
slot 14 (0,4) (2,1) (3,17) (6,13) (7,14) (10,9) (12,11) (15,5) (16,8)
slot 15 (1,6) (2,12) (4,16) (5,10) (7,3) (8,15) (9,11) (13,0) (17,14)
slot 16 (0,1) (4,17) (6,16) (9,8) (10,3) (11,2) (12,5) (14,13) (15,7)
slot 17 (1,12) (2,6) (3,11) (5,4) (7,8) (10,15) (13,9) (16,14) (17,0)
slot 18 (0,11) (2,16) (6,9) (10,7) (12,4) (13,5) (14,1) (15,3) (17,8)
slot 19 (1,3) (2,17) (4,0) (6,7) (8,5) (9,15) (13,11) (14,10) (16,12)
slot 20 (3,0) (5,16) (6,8) (7,2) (11,10) (12,9) (13,1) (14,4) (15,17)
slot 21 (1,7) (2,15) (4,6) (8,12) (9,0) (10,13) (11,14) (16,3) (17,5)
slot 22 (0,16) (3,9) (4,11) (5,2) (6,17) (12,10) (13,7) (14,8) (15,1)
slot 23 (1,5) (3,2) (7,4) (8,0) (9,14) (10,16) (11,6) (15,12) (17,13)
slot 24 (1,8) (2,0) (4,3) (5,11) (6,10) (7,17) (12,14) (13,15) (16,9)
slot 25 (0,7) (3,8) (9,2) (10,4) (11,1) (12,13) (14,5) (15,6) (17,16)
slot 26 (0,10) (1,17) (5,9) (6,3) (7,12) (8,4) (13,2) (14,15) (16,11)
slot 27 (2,14) (3,7) (4,13) (5,0) (9,1) (10,8) (11,15) (12,17) (16,6)
slot 28 (0,6) (4,1) (8,11) (10,5) (12,2) (13,3) (14,7) (15,16) (17,9)

slot 29 (0,12) (1,2) (3,10) (5,15) (6,14) (9,4) (11,17) (13,8) (16,7)
slot 30 (2,11) (7,5) (8,9) (10,1) (12,6) (14,3) (15,0) (16,13) (17,4)
slot 31 (0,14) (1,16) (4,15) (5,12) (7,11) (8,2) (9,10) (13,6) (17,3)
slot 32 (0,13) (2,10) (3,5) (6,1) (9,7) (11,12) (14,17) (15,8) (16,4)
slot 33 (1,0) (4,2) (5,6) (7,15) (8,16) (10,17) (11,9) (12,3) (13,14)

Πρόβλημα Late 7

Βρήκαμε την λύση (0, 7082) που έχει ανάλυση κόστους

CA1: 26
CA2: 1610
CA3: 445
GA1: 21
SE1: 4980

και ωρολόγιο πρόγραμμα:

slot 0 (1,15) (3,11) (4,7) (5,9) (6,10) (12,2) (13,16) (14,0) (17,8)
slot 1 (0,16) (2,3) (7,6) (8,12) (9,17) (10,14) (11,5) (13,4) (15,1)
slot 2 (1,0) (2,13) (3,6) (4,9) (5,10) (12,11) (14,8) (16,7) (17,15)
slot 3 (0,11) (1,3) (6,17) (7,12) (8,4) (9,2) (10,16) (13,14) (15,5)
slot 4 (4,5) (8,2) (9,1) (11,6) (12,0) (14,10) (15,3) (16,13) (17,7)
slot 5 (0,12) (1,11) (2,4) (3,5) (6,15) (7,8) (10,17) (13,9) (16,14)
slot 6 (2,16) (3,14) (5,6) (8,7) (9,10) (11,0) (12,13) (15,4) (17,1)
slot 7 (0,9) (4,2) (6,8) (7,3) (10,15) (13,11) (14,5) (16,1) (17,12)
slot 8 (0,5) (1,13) (2,14) (3,4) (8,17) (9,6) (11,10) (12,7) (15,16)
slot 9 (4,0) (5,8) (6,9) (7,15) (10,2) (13,12) (14,1) (16,11) (17,3)
slot 10 (0,7) (3,2) (5,16) (8,13) (9,4) (11,14) (12,1) (15,6) (17,10)
slot 11 (2,0) (4,1) (6,13) (7,17) (8,3) (9,12) (10,11) (14,15) (16,5)
slot 12 (0,6) (1,2) (3,10) (5,4) (11,9) (12,16) (13,8) (15,7) (17,14)
slot 13 (2,6) (3,15) (4,17) (7,1) (8,5) (9,13) (10,0) (14,11) (16,12)
slot 14 (1,16) (5,0) (6,3) (8,14) (11,2) (12,10) (13,7) (15,9) (17,4)
slot 15 (0,1) (2,5) (3,13) (4,6) (9,8) (10,7) (11,15) (14,12) (16,17)
slot 16 (1,4) (5,2) (6,12) (7,10) (8,11) (13,17) (14,9) (15,0) (16,3)
slot 17 (2,15) (3,16) (4,8) (5,13) (6,0) (9,7) (10,1) (11,17) (12,14)
slot 18 (0,4) (1,12) (7,11) (8,9) (13,6) (14,3) (15,10) (16,2) (17,5)
slot 19 (2,1) (3,17) (4,16) (5,15) (7,14) (9,0) (10,6) (11,13) (12,8)
slot 20 (0,3) (1,7) (5,12) (6,4) (8,10) (13,2) (14,16) (15,11) (17,9)
slot 21 (2,17) (3,8) (4,14) (7,0) (9,15) (10,12) (11,1) (13,5) (16,6)

slot 22 (0,13) (1,9) (6,2) (10,4) (11,3) (12,5) (14,7) (15,8) (17,16)
slot 23 (2,10) (3,1) (5,14) (7,4) (8,6) (9,11) (12,17) (13,0) (16,15)
slot 24 (0,8) (1,5) (3,12) (4,15) (6,16) (10,9) (11,7) (14,2) (17,13)
slot 25 (0,10) (2,11) (5,3) (6,7) (8,15) (12,4) (13,1) (14,17) (16,9)
slot 26 (1,6) (4,3) (7,16) (8,0) (9,5) (10,13) (11,12) (15,14) (17,2)
slot 27 (0,17) (1,10) (2,7) (3,9) (4,12) (5,11) (6,14) (13,15) (16,8)
slot 28 (6,11) (7,13) (8,1) (9,16) (10,5) (12,3) (14,4) (15,2) (17,0)
slot 29 (0,15) (1,14) (2,9) (3,7) (4,13) (5,17) (11,8) (12,6) (16,10)
slot 30 (0,2) (6,1) (7,5) (9,14) (10,8) (13,3) (15,12) (16,4) (17,11)
slot 31 (1,17) (2,8) (3,0) (4,10) (5,7) (11,16) (12,9) (14,6) (15,13)
slot 32 (0,14) (4,11) (5,1) (7,2) (8,16) (9,3) (12,15) (13,10) (17,6)
slot 33 (1,8) (2,12) (6,5) (7,9) (10,3) (11,4) (14,13) (15,17) (16,0)

Πρόβλημα Late 8

Βρήκαμε την λύση (0, 1971) που έχει ανάλυση κόστους

CA1: 8
CA2: 30
CA3: 490
GA1: 13
BR2: 1300
SE1: 130

και ωρολόγιο πρόγραμμα:

slot 0 (1,3) (2,8) (4,7) (9,12) (11,0) (14,10) (15,6) (16,5) (17,13)
slot 1 (0,15) (3,2) (4,14) (5,10) (6,11) (7,9) (8,1) (12,17) (13,16)
slot 2 (0,8) (1,15) (2,16) (5,12) (6,3) (9,17) (10,4) (13,11) (14,7)
slot 3 (1,0) (4,16) (7,5) (8,3) (10,9) (11,2) (12,13) (15,14) (17,6)
slot 4 (2,6) (3,12) (8,4) (9,11) (13,10) (14,0) (15,5) (16,1) (17,7)
slot 5 (1,4) (3,15) (5,14) (6,13) (7,10) (9,0) (11,8) (12,2) (17,16)
slot 6 (1,17) (2,0) (4,12) (5,3) (8,7) (10,11) (13,15) (14,6) (16,9)
slot 7 (2,14) (4,5) (6,1) (7,0) (9,13) (10,8) (12,16) (15,11) (17,3)
slot 8 (0,12) (1,2) (3,9) (5,17) (6,10) (8,14) (11,16) (13,7) (15,4)
slot 9 (0,13) (2,4) (9,5) (10,15) (11,7) (12,6) (14,1) (16,3) (17,8)
slot 10 (3,10) (4,0) (5,2) (6,7) (8,9) (12,14) (13,1) (15,16) (17,11)
slot 11 (0,3) (1,12) (2,9) (4,13) (6,5) (7,15) (10,17) (11,14) (16,8)
slot 12 (3,4) (5,0) (7,2) (8,13) (9,15) (10,1) (12,11) (16,6) (17,14)
slot 13 (1,9) (2,10) (6,8) (11,4) (12,7) (13,5) (14,3) (15,17) (16,0)

slot 14 (0,17) (2,13) (3,7) (4,6) (8,5) (9,14) (10,16) (11,1) (12,15)
slot 15 (6,9) (7,1) (8,12) (10,0) (11,5) (13,3) (14,16) (15,2) (17,4)
slot 16 (0,6) (1,5) (2,17) (3,11) (4,9) (8,15) (12,10) (14,13) (16,7)
slot 17 (5,16) (7,14) (8,2) (9,3) (11,10) (12,4) (13,6) (15,0) (17,1)
slot 18 (0,11) (1,8) (2,12) (3,5) (6,17) (9,7) (10,13) (14,15) (16,4)
slot 19 (0,14) (4,2) (5,6) (7,11) (8,10) (9,16) (12,3) (13,17) (15,1)
slot 20 (0,7) (3,8) (4,1) (6,12) (10,5) (11,13) (14,2) (16,15) (17,9)
slot 21 (1,6) (5,4) (7,13) (9,8) (12,0) (14,11) (15,3) (16,2) (17,10)
slot 22 (3,1) (5,7) (6,15) (8,0) (9,2) (10,14) (11,17) (13,4) (16,12)
slot 23 (0,16) (1,11) (2,3) (4,10) (6,14) (7,8) (13,12) (15,9) (17,5)
slot 24 (3,0) (5,15) (7,17) (8,6) (9,1) (10,2) (11,12) (14,4) (16,13)
slot 25 (1,14) (2,5) (4,11) (6,16) (7,3) (12,9) (13,8) (15,10) (17,0)
slot 26 (0,1) (2,15) (3,13) (4,17) (5,11) (8,16) (9,6) (10,7) (14,12)
slot 27 (0,2) (1,10) (4,3) (5,9) (7,16) (11,6) (12,8) (13,14) (17,15)
slot 28 (0,5) (2,1) (3,6) (7,4) (8,11) (10,12) (14,9) (15,13) (16,17)
slot 29 (1,7) (4,15) (5,8) (6,2) (9,10) (11,3) (13,0) (16,14) (17,12)
slot 30 (0,4) (1,16) (2,11) (3,17) (10,6) (12,5) (13,9) (14,8) (15,7)
slot 31 (3,14) (5,1) (6,0) (7,12) (8,17) (9,4) (11,15) (13,2) (16,10)
slot 32 (0,9) (1,13) (4,8) (7,6) (10,3) (14,5) (15,12) (16,11) (17,2)
slot 33 (0,10) (2,7) (3,16) (5,13) (6,4) (11,9) (12,1) (14,17) (15,8)

Πρόβλημα Late 9

Βρήκαμε την λύση (0, 2426) που έχει ανάλυση κόστους

CA2: 640
CA3: 520
GA1: 16
BR1: 10
BR2: 1240

και ωρολόγιο πρόγραμμα:

slot 0 (2,16) (3,5) (7,6) (8,9) (10,4) (11,14) (12,17) (13,0) (15,1)
slot 1 (0,5) (1,15) (3,2) (6,8) (9,4) (13,12) (14,7) (16,10) (17,11)
slot 2 (0,1) (4,3) (5,17) (6,13) (7,2) (8,11) (9,16) (10,15) (12,14)
slot 3 (2,17) (3,6) (4,1) (8,7) (10,12) (11,9) (14,0) (15,13) (16,5)
slot 4 (0,11) (1,14) (5,8) (6,10) (7,15) (9,3) (12,2) (13,4) (17,16)
slot 5 (3,9) (5,0) (8,13) (10,1) (11,7) (14,4) (15,12) (16,2) (17,6)
slot 6 (1,0) (2,10) (4,8) (6,7) (9,17) (12,11) (13,5) (15,14) (16,3)

slot 7 (0,8) (2,3) (5,12) (7,4) (9,15) (10,6) (11,1) (14,16) (17,13)
slot 8 (1,12) (3,10) (4,7) (6,9) (8,0) (11,2) (13,15) (14,5) (16,17)
slot 9 (1,8) (4,9) (5,2) (6,3) (7,0) (10,17) (12,16) (13,14) (15,11)
slot 10 (0,10) (2,4) (3,15) (5,13) (9,7) (11,6) (14,8) (16,1) (17,12)
slot 11 (2,13) (4,10) (7,1) (8,14) (9,0) (12,3) (15,6) (16,11) (17,5)
slot 12 (0,3) (1,17) (5,4) (6,2) (10,14) (11,16) (12,8) (13,9) (15,7)
slot 13 (2,6) (3,0) (4,15) (5,11) (7,9) (8,10) (13,16) (14,12) (17,1)
slot 14 (0,4) (1,13) (3,12) (8,17) (9,6) (10,11) (14,2) (15,5) (16,7)
slot 15 (0,9) (2,8) (5,14) (6,1) (7,17) (11,4) (12,10) (13,3) (16,15)
slot 16 (1,5) (3,11) (4,13) (6,14) (7,8) (9,12) (10,16) (15,2) (17,0)
slot 17 (0,16) (2,1) (5,3) (8,6) (11,10) (12,4) (13,7) (14,15) (17,9)
slot 18 (1,6) (3,17) (4,14) (7,12) (9,2) (10,0) (11,5) (15,8) (16,13)
slot 19 (0,12) (2,9) (3,1) (4,16) (5,7) (6,11) (8,15) (13,10) (17,14)
slot 20 (1,4) (9,8) (10,5) (11,3) (12,7) (14,13) (15,0) (16,6) (17,2)
slot 21 (0,15) (1,3) (2,14) (4,5) (6,17) (7,16) (8,12) (9,11) (10,13)
slot 22 (0,2) (3,7) (4,6) (5,9) (11,8) (12,1) (13,17) (14,10) (15,16)
slot 23 (1,7) (2,0) (6,4) (8,5) (10,3) (11,13) (14,9) (16,12) (17,15)
slot 24 (0,14) (1,2) (4,11) (6,5) (7,3) (9,10) (12,13) (15,17) (16,8)
slot 25 (0,17) (2,7) (3,16) (5,15) (8,4) (10,9) (11,12) (13,6) (14,1)
slot 26 (2,5) (3,8) (6,16) (7,13) (9,1) (12,0) (14,11) (15,4) (17,10)
slot 27 (0,6) (1,9) (4,2) (5,10) (8,3) (12,15) (13,11) (16,14) (17,7)
slot 28 (1,16) (2,12) (4,17) (6,0) (7,5) (9,13) (10,8) (11,15) (14,3)
slot 29 (0,13) (3,14) (5,1) (7,11) (8,2) (12,6) (15,10) (16,9) (17,4)
slot 30 (2,11) (4,0) (5,6) (8,16) (10,7) (12,9) (13,1) (14,17) (15,3)
slot 31 (1,10) (2,15) (6,12) (7,14) (9,5) (11,0) (13,8) (16,4) (17,3)
slot 32 (0,7) (3,13) (4,12) (5,16) (6,15) (8,1) (9,14) (10,2) (11,17)
slot 33 (1,11) (3,4) (7,10) (12,5) (13,2) (14,6) (15,9) (16,0) (17,8)

Πρόβλημα Late 10

Το αποτέλεσμα που βρήκαμε (21, 3997), το οποίο δεν είναι εφικτή λύση και έχει ανάλυση κόστους

Ισχυροί περιορισμοί
233 = 219 ικανοποιούνται + 14 παραβιάζονται
CA4: 9
GA1: 4
BR2: 8

Ασθενείς περιορισμοί

CA1: 36

CA2: 1140

CA4: 1355

GA1: 6

SE1: 1460

και ωρολόγιο πρόγραμμα:

slot 0 (4,12) (5,2) (7,6) (8,11) (9,3) (13,14) (16,0) (17,10) (18,15) (19,1)
slot 1 (0,18) (1,14) (5,9) (6,19) (7,8) (10,2) (11,13) (12,17) (15,3) (16,4)
slot 2 (2,15) (3,11) (4,8) (9,7) (12,10) (13,1) (14,16) (17,0) (18,6) (19,5)
slot 3 (0,5) (1,9) (4,14) (6,3) (7,18) (8,17) (11,19) (13,2) (15,12) (16,10)
slot 4 (1,8) (2,0) (3,16) (5,15) (7,13) (9,11) (10,14) (12,6) (18,4) (19,17)
slot 5 (0,6) (4,2) (8,10) (11,5) (12,18) (13,3) (14,9) (15,19) (16,7) (17,1)
slot 6 (1,5) (2,11) (3,12) (4,17) (6,16) (9,15) (10,7) (14,0) (18,13) (19,8)
slot 7 (0,4) (2,3) (5,6) (7,19) (8,15) (9,18) (11,16) (12,1) (13,10) (17,14)
slot 8 (1,2) (3,4) (6,9) (7,5) (8,12) (10,11) (13,17) (15,14) (16,18) (19,0)
slot 9 (1,0) (2,19) (4,10) (5,8) (9,16) (11,7) (12,13) (14,6) (17,15) (18,3)
slot 10 (0,12) (3,5) (6,1) (7,17) (9,2) (14,8) (15,11) (16,13) (18,10) (19,4)
slot 11 (0,7) (1,3) (2,14) (5,4) (8,9) (10,6) (11,18) (12,19) (13,15) (17,16)
slot 12 (3,17) (4,11) (6,8) (9,12) (13,0) (14,7) (15,1) (16,5) (18,2) (19,10)
slot 13 (0,15) (1,4) (2,12) (5,13) (7,3) (10,9) (11,14) (16,8) (17,6) (18,19)
slot 14 (1,18) (3,0) (6,11) (8,13) (9,4) (10,5) (12,7) (14,19) (15,16) (17,2)
slot 15 (0,10) (2,8) (4,6) (5,14) (7,15) (11,1) (13,9) (16,12) (18,17) (19,3)
slot 16 (1,7) (2,16) (5,17) (6,13) (8,0) (9,19) (10,3) (11,12) (14,18) (15,4)
slot 17 (0,9) (3,8) (6,15) (7,2) (10,1) (12,14) (13,4) (16,19) (17,11) (18,5)
slot 18 (1,16) (2,6) (4,7) (5,12) (8,18) (9,17) (11,0) (14,3) (15,10) (19,13)
slot 19 (0,11) (3,14) (6,2) (7,1) (8,4) (10,19) (12,15) (13,5) (16,9) (17,18)
slot 20 (1,10) (3,6) (4,9) (5,0) (11,2) (14,13) (15,8) (16,17) (18,7) (19,12)
slot 21 (0,16) (2,18) (6,14) (7,4) (8,5) (9,1) (10,15) (12,3) (17,13) (19,11)
slot 22 (3,10) (4,1) (5,7) (12,0) (13,19) (14,11) (15,6) (16,2) (17,8) (18,9)
slot 23 (0,13) (1,15) (2,4) (6,17) (7,16) (8,14) (9,5) (10,12) (11,3) (19,18)
slot 24 (0,8) (3,2) (4,19) (5,10) (12,11) (13,6) (14,1) (15,7) (17,9) (18,16)
slot 25 (1,12) (2,9) (5,18) (7,0) (8,3) (10,13) (11,4) (15,17) (16,14) (19,6)
slot 26 (0,19) (3,1) (4,5) (6,10) (9,8) (12,2) (13,16) (14,15) (17,7) (18,11)
slot 27 (2,13) (5,3) (7,12) (8,6) (10,18) (11,9) (15,0) (16,1) (17,4) (19,14)
slot 28 (0,17) (1,13) (3,19) (4,16) (6,7) (9,10) (11,15) (12,5) (14,2) (18,8)
slot 29 (0,3) (6,18) (7,14) (8,1) (10,4) (13,12) (15,9) (16,11) (17,5) (19,2)
slot 30 (1,6) (2,17) (3,15) (4,13) (5,16) (9,0) (11,8) (14,10) (18,12) (19,7)

slot 31 (0,14) (1,19) (6,4) (7,11) (8,2) (10,17) (12,9) (13,18) (15,5) (16,3)
slot 32 (2,7) (3,13) (4,15) (5,1) (8,16) (9,6) (11,10) (14,12) (17,19) (18,0)
slot 33 (1,17) (3,18) (6,5) (7,9) (10,0) (12,8) (13,11) (14,4) (15,2) (19,16)
slot 34 (0,1) (2,10) (4,3) (5,19) (8,7) (9,13) (11,6) (16,15) (17,12) (18,14)
slot 35 (1,11) (2,5) (3,7) (4,0) (6,12) (10,16) (13,8) (14,17) (15,18) (19,9)
slot 36 (0,2) (5,11) (7,10) (8,19) (9,14) (12,4) (15,13) (16,6) (17,3) (18,1)
slot 37 (2,1) (3,9) (4,18) (6,0) (10,8) (11,17) (12,16) (13,7) (14,5) (19,15)

Πρόβλημα Late 11

Βρήκαμε την λύση (0, 796) που έχει ανάλυση κόστους

CA2: 610
CA3: 185
GA1: 1

και ωρολόγιο πρόγραμμα:

slot 0 (0,13) (2,4) (5,3) (8,7) (9,6) (10,18) (12,11) (15,1) (16,17) (19,14)
slot 1 (1,0) (3,15) (4,16) (6,12) (7,5) (11,10) (13,2) (14,8) (17,19) (18,9)
slot 2 (2,18) (4,11) (5,17) (7,3) (8,6) (9,15) (12,0) (14,10) (16,13) (19,1)
slot 3 (0,7) (1,14) (3,8) (4,9) (6,2) (10,16) (11,19) (13,17) (15,12) (18,5)
slot 4 (2,3) (5,0) (7,15) (9,13) (10,6) (12,8) (14,4) (16,11) (17,1) (19,18)
slot 5 (0,10) (1,13) (3,19) (4,7) (6,17) (8,9) (11,18) (12,5) (15,2) (16,14)
slot 6 (0,9) (2,1) (5,16) (7,19) (10,8) (13,6) (14,11) (15,4) (17,12) (18,3)
slot 7 (2,17) (3,10) (4,13) (6,15) (8,18) (9,14) (11,7) (12,1) (16,0) (19,5)
slot 8 (0,2) (1,6) (5,11) (7,12) (13,8) (14,3) (15,10) (17,9) (18,16) (19,4)
slot 9 (2,7) (4,18) (6,5) (8,1) (9,19) (10,17) (11,0) (12,13) (14,15) (16,3)
slot 10 (1,16) (3,11) (5,2) (6,4) (7,9) (13,10) (15,0) (17,8) (18,14) (19,12)
slot 11 (0,19) (3,13) (4,1) (8,15) (10,2) (11,17) (12,9) (14,5) (16,6) (18,7)
slot 12 (0,18) (1,10) (2,11) (7,6) (8,5) (9,16) (12,4) (13,14) (15,19) (17,3)
slot 13 (3,12) (4,10) (5,1) (6,0) (7,13) (11,9) (14,17) (16,8) (18,15) (19,2)
slot 14 (0,14) (1,18) (6,19) (8,2) (9,3) (10,7) (12,16) (13,5) (15,11) (17,4)
slot 15 (2,16) (3,6) (4,0) (5,9) (7,1) (11,8) (14,12) (15,17) (18,13) (19,10)
slot 16 (1,11) (3,4) (6,14) (8,0) (9,2) (10,5) (12,18) (13,19) (16,15) (17,7)
slot 17 (0,3) (1,9) (2,14) (4,5) (7,16) (10,12) (11,6) (15,13) (18,17) (19,8)
slot 18 (3,1) (5,15) (6,18) (8,4) (9,10) (12,2) (13,11) (14,7) (16,19) (17,0)
slot 19 (0,16) (1,3) (2,9) (4,19) (6,8) (7,17) (11,14) (13,12) (15,5) (18,10)
slot 20 (3,2) (4,6) (5,8) (9,1) (10,11) (12,15) (14,18) (16,7) (17,13) (19,0)
slot 21 (0,17) (1,12) (2,5) (6,16) (7,10) (8,14) (11,3) (13,15) (18,4) (19,9)

slot 22 (5,7) (8,19) (9,11) (10,0) (13,4) (14,1) (15,3) (16,2) (17,6) (18,12)
slot 23 (1,5) (2,19) (3,0) (4,17) (6,13) (7,14) (9,8) (11,15) (12,10) (16,18)
slot 24 (0,12) (5,6) (8,11) (10,4) (13,7) (14,9) (15,16) (17,2) (18,1) (19,3)
slot 25 (1,8) (2,0) (3,5) (6,10) (7,18) (9,17) (11,13) (12,14) (16,4) (19,15)
slot 26 (0,11) (4,3) (5,19) (8,12) (10,1) (13,9) (14,6) (15,7) (17,16) (18,2)
slot 27 (0,5) (1,7) (2,8) (3,18) (6,9) (11,4) (12,17) (15,14) (16,10) (19,13)
slot 28 (1,19) (4,2) (7,11) (8,3) (9,12) (10,15) (13,0) (14,16) (17,5) (18,6)
slot 29 (0,8) (2,10) (3,9) (5,4) (12,7) (13,18) (15,6) (16,1) (17,14) (19,11)
slot 30 (1,17) (4,15) (6,3) (7,2) (8,16) (9,5) (10,19) (11,12) (14,13) (18,0)
slot 31 (0,4) (2,6) (3,16) (5,14) (7,8) (12,19) (13,1) (15,9) (17,10) (18,11)
slot 32 (1,15) (5,18) (6,7) (8,13) (9,4) (10,3) (11,2) (14,0) (16,12) (19,17)
slot 33 (0,1) (2,13) (3,7) (4,12) (10,14) (11,5) (16,9) (17,15) (18,8) (19,6)
slot 34 (1,2) (5,10) (6,11) (7,4) (8,17) (9,0) (12,3) (13,16) (14,19) (15,18)
slot 35 (0,6) (2,15) (3,14) (4,8) (5,12) (9,7) (10,13) (11,1) (17,18) (19,16)
slot 36 (1,4) (7,0) (10,9) (12,6) (13,3) (14,2) (15,8) (16,5) (17,11) (18,19)
slot 37 (0,15) (2,12) (3,17) (4,14) (5,13) (6,1) (8,10) (9,18) (11,16) (19,7)

Πρόβλημα Late 12

Βρήκαμε την λύση (0, 9566) που έχει ανάλυση κόστους

CA1: 26
CA2: 1635
CA3: 140
CA4: 2115
SE1: 5650

και ωρολόγιο πρόγραμμα:

slot 0 (0,9) (2,10) (3,14) (8,7) (12,4) (13,16) (15,1) (17,6) (18,5) (19,11)
slot 1 (1,13) (4,3) (5,8) (6,2) (7,0) (9,15) (10,18) (11,17) (14,19) (16,12)
slot 2 (0,7) (3,6) (8,14) (9,4) (11,19) (12,5) (13,1) (15,16) (17,2) (18,10)
slot 3 (1,3) (2,13) (5,12) (6,0) (7,18) (8,4) (10,11) (16,14) (17,15) (19,9)
slot 4 (0,1) (4,2) (7,8) (9,19) (11,6) (12,3) (13,17) (14,5) (16,10) (18,15)
slot 5 (1,8) (2,16) (3,0) (5,13) (6,14) (10,17) (11,7) (12,9) (15,18) (19,4)
slot 6 (0,2) (1,19) (4,6) (5,10) (7,9) (8,11) (13,18) (14,12) (16,15) (17,3)
slot 7 (2,0) (3,7) (4,8) (9,5) (10,6) (11,14) (12,16) (15,13) (18,1) (19,17)
slot 8 (0,16) (1,4) (3,8) (5,2) (6,18) (7,13) (9,12) (11,15) (14,10) (17,19)
slot 9 (0,14) (2,5) (4,17) (8,1) (10,15) (12,11) (13,3) (16,6) (18,9) (19,7)
slot 10 (2,1) (3,17) (5,16) (6,12) (7,10) (8,19) (9,13) (11,4) (14,18) (15,0)

slot 11 (0,3) (1,2) (4,9) (10,14) (13,7) (15,6) (16,11) (17,12) (18,8) (19,5)
slot 12 (0,11) (2,17) (3,1) (5,15) (6,9) (7,19) (8,10) (12,18) (14,4) (16,13)
slot 13 (1,16) (4,10) (7,2) (9,3) (11,0) (13,14) (15,12) (17,5) (18,6) (19,8)
slot 14 (0,18) (2,4) (5,1) (6,11) (10,8) (12,13) (14,3) (16,9) (17,7) (19,15)
slot 15 (1,5) (3,19) (4,16) (7,17) (8,0) (9,14) (10,2) (13,6) (15,11) (18,12)
slot 16 (2,15) (5,18) (6,4) (11,1) (12,0) (13,9) (14,8) (16,7) (17,10) (19,3)
slot 17 (0,19) (1,12) (3,9) (7,11) (8,6) (10,4) (15,14) (16,5) (17,13) (18,2)
slot 18 (2,19) (3,18) (4,0) (5,7) (6,17) (9,8) (11,10) (12,1) (13,15) (14,16)
slot 19 (0,5) (1,14) (6,3) (7,4) (8,2) (10,16) (15,9) (17,11) (18,13) (19,12)
slot 20 (2,6) (3,10) (4,1) (5,11) (9,7) (12,17) (13,8) (14,0) (15,19) (16,18)
slot 21 (0,4) (2,9) (6,15) (7,3) (8,12) (10,13) (11,5) (17,16) (18,14) (19,1)
slot 22 (1,10) (3,2) (4,11) (9,0) (12,7) (13,5) (14,6) (15,17) (16,8) (18,19)
slot 23 (0,13) (2,14) (4,18) (5,6) (7,15) (8,3) (10,9) (11,12) (17,1) (19,16)
slot 24 (1,0) (3,4) (6,7) (8,18) (9,11) (10,19) (13,12) (14,2) (15,5) (16,17)
slot 25 (0,10) (1,6) (2,8) (4,14) (5,3) (7,16) (11,9) (12,15) (18,17) (19,13)
slot 26 (3,12) (6,19) (8,15) (9,1) (10,5) (11,18) (13,0) (14,7) (16,2) (17,4)
slot 27 (0,6) (1,11) (4,13) (5,9) (7,14) (8,17) (12,10) (15,3) (18,16) (19,2)
slot 28 (2,11) (3,5) (6,8) (9,18) (10,12) (13,4) (14,1) (15,7) (16,19) (17,0)
slot 29 (1,17) (3,13) (4,15) (5,14) (8,16) (9,10) (11,2) (12,6) (18,7) (19,0)
slot 30 (0,12) (2,3) (6,1) (7,5) (13,11) (14,9) (15,10) (16,4) (17,8) (19,18)
slot 31 (1,15) (4,19) (5,17) (6,16) (8,13) (9,2) (10,7) (11,3) (12,14) (18,0)
slot 32 (0,8) (2,18) (3,11) (4,5) (7,12) (13,10) (14,15) (16,1) (17,9) (19,6)
slot 33 (1,7) (5,19) (6,13) (9,16) (10,0) (11,8) (12,2) (15,4) (17,14) (18,3)
slot 34 (0,17) (1,18) (2,7) (3,15) (4,12) (6,5) (8,9) (11,16) (14,13) (19,10)
slot 35 (5,0) (7,6) (9,17) (10,1) (12,19) (13,2) (14,11) (15,8) (16,3) (18,4)
slot 36 (0,15) (1,9) (2,12) (3,16) (4,7) (6,10) (8,5) (11,13) (17,18) (19,14)
slot 37 (5,4) (7,1) (9,6) (10,3) (12,8) (13,19) (14,17) (15,2) (16,0) (18,11)

Πρόβλημα Late 13

Βρήκαμε την λύση (0, 7810) που έχει ανάλυση κόστους

CA1: 21
CA2: 50
CA4: 475
GA1: 4
BR2: 1800
FA2: 160

SE1: 5300

και ωρολόγιο πρόγραμμα:

slot 0 (0,18) (2,13) (3,17) (4,15) (6,1) (7,19) (9,10) (11,8) (12,16) (14,5)
slot 1 (1,12) (3,2) (5,8) (7,11) (10,9) (13,6) (15,14) (16,4) (17,0) (19,18)
slot 2 (0,3) (2,19) (4,17) (5,1) (6,10) (8,15) (9,16) (12,14) (13,11) (18,7)
slot 3 (0,4) (1,18) (3,6) (8,2) (10,19) (11,5) (14,9) (15,7) (16,13) (17,12)
slot 4 (2,16) (3,11) (4,10) (6,18) (7,12) (9,14) (13,8) (15,0) (17,1) (19,5)
slot 5 (0,9) (1,4) (5,7) (6,16) (10,15) (11,19) (12,8) (13,17) (14,2) (18,3)
slot 6 (2,4) (3,5) (7,9) (8,17) (11,13) (12,6) (15,1) (16,10) (18,14) (19,0)
slot 7 (1,2) (4,18) (5,3) (8,12) (9,0) (10,16) (13,7) (14,11) (15,19) (17,6)
slot 8 (0,10) (1,16) (2,9) (6,5) (7,13) (11,3) (12,17) (14,15) (18,4) (19,8)
slot 9 (0,11) (3,15) (5,10) (7,1) (8,4) (9,2) (13,19) (16,12) (17,14) (18,6)
slot 10 (2,1) (4,11) (8,0) (9,6) (10,7) (12,5) (14,13) (15,18) (16,17) (19,3)
slot 11 (0,19) (1,9) (3,16) (6,8) (7,14) (10,12) (11,4) (15,5) (17,13) (18,2)
slot 12 (0,1) (4,6) (5,2) (8,11) (9,3) (12,10) (13,15) (14,7) (16,18) (17,19)
slot 13 (1,8) (2,12) (3,0) (5,14) (6,15) (7,17) (10,11) (13,16) (18,9) (19,4)
slot 14 (4,14) (7,16) (8,3) (9,13) (10,5) (11,2) (12,0) (15,17) (18,1) (19,6)
slot 15 (0,5) (2,7) (3,12) (6,4) (8,18) (13,1) (14,10) (15,9) (16,19) (17,11)
slot 16 (1,15) (2,8) (3,18) (4,19) (5,16) (6,14) (7,0) (10,13) (11,17) (12,9)
slot 17 (0,16) (4,1) (9,8) (11,10) (13,12) (14,6) (15,2) (17,3) (18,5) (19,7)
slot 18 (1,0) (3,9) (5,13) (6,11) (8,10) (12,18) (14,4) (16,7) (17,2) (19,15)
slot 19 (0,12) (2,11) (3,14) (4,13) (5,9) (6,19) (7,15) (10,8) (16,1) (18,17)
slot 20 (1,6) (2,0) (8,5) (9,4) (11,12) (13,3) (14,18) (15,16) (17,7) (19,10)
slot 21 (3,13) (4,2) (5,19) (9,7) (11,0) (12,1) (14,17) (15,8) (16,6) (18,10)
slot 22 (0,14) (1,3) (4,5) (6,13) (7,8) (10,17) (12,2) (16,9) (18,15) (19,11)
slot 23 (2,18) (5,12) (6,9) (7,3) (8,1) (10,0) (13,4) (14,19) (15,11) (17,16)
slot 24 (0,8) (1,10) (3,7) (9,5) (11,14) (12,4) (15,6) (16,2) (18,13) (19,17)
slot 25 (1,13) (2,10) (4,12) (5,17) (6,0) (8,7) (9,15) (14,3) (18,11) (19,16)
slot 26 (4,0) (5,15) (7,2) (8,9) (10,6) (11,1) (12,19) (13,14) (16,3) (17,18)
slot 27 (0,2) (3,1) (6,12) (7,10) (9,11) (15,13) (16,5) (17,4) (18,8) (19,14)
slot 28 (1,17) (2,5) (3,19) (4,9) (6,7) (8,16) (11,15) (13,10) (14,12) (18,0)
slot 29 (5,4) (7,6) (10,2) (12,11) (13,18) (14,8) (15,3) (16,0) (17,9) (19,1)
slot 30 (0,15) (1,7) (2,17) (4,3) (5,11) (8,6) (9,19) (10,14) (12,13) (18,16)
slot 31 (3,10) (6,2) (7,4) (9,1) (11,18) (13,5) (14,0) (16,15) (17,8) (19,12)
slot 32 (0,6) (1,19) (2,3) (4,16) (8,14) (10,18) (11,7) (13,9) (15,12) (17,5)
slot 33 (0,13) (2,15) (4,8) (5,6) (9,17) (10,3) (12,7) (14,1) (16,11) (18,19)
slot 34 (1,14) (3,4) (5,0) (6,17) (7,18) (8,13) (9,12) (11,16) (15,10) (19,2)
slot 35 (1,11) (2,6) (3,8) (7,5) (13,0) (15,4) (16,14) (17,10) (18,12) (19,9)

slot 36 (0,17) (4,7) (5,18) (6,3) (8,19) (10,1) (11,9) (12,15) (13,2) (14,16)
slot 37 (0,7) (1,5) (2,14) (9,18) (10,4) (11,6) (12,3) (16,8) (17,15) (19,13)

Πρόβλημα Late 14

Βρήκαμε την λύση (0, 1966) που έχει ανάλυση κόστους

CA2: 600
CA3: 230
CA4: 1075
GA1: 51
BR1: 10

και ωρολόγιο πρόγραμμα:

slot 0 (0,12) (1,8) (2,11) (4,19) (5,17) (9,6) (13,18) (14,3) (15,7) (16,10)
slot 1 (3,16) (5,0) (6,14) (7,1) (8,4) (9,15) (10,19) (11,17) (12,18) (13,2)
slot 2 (0,16) (1,13) (3,2) (4,11) (8,7) (12,9) (15,6) (17,10) (18,14) (19,5)
slot 3 (2,3) (5,15) (6,8) (7,18) (9,12) (10,13) (11,1) (14,19) (16,0) (17,4)
slot 4 (0,11) (1,4) (2,15) (7,9) (10,12) (13,5) (14,6) (16,8) (18,3) (19,17)
slot 5 (1,14) (3,13) (4,18) (5,11) (6,10) (8,19) (9,16) (12,7) (15,0) (17,2)
slot 6 (2,10) (4,12) (7,3) (8,1) (11,15) (13,6) (14,5) (17,0) (18,16) (19,9)
slot 7 (0,1) (3,7) (5,2) (9,14) (10,17) (11,6) (12,4) (16,13) (18,15) (19,8)
slot 8 (0,13) (1,11) (3,10) (4,5) (6,16) (7,19) (8,18) (12,2) (15,14) (17,9)
slot 9 (1,0) (2,7) (6,5) (9,4) (10,8) (11,3) (13,12) (14,16) (15,17) (18,19)
slot 10 (2,13) (3,8) (4,15) (5,14) (9,0) (10,18) (12,1) (16,7) (17,11) (19,6)
slot 11 (0,5) (1,6) (7,4) (8,10) (11,19) (13,17) (14,12) (15,3) (16,9) (18,2)
slot 12 (4,0) (5,1) (6,2) (10,9) (11,13) (12,3) (14,7) (15,8) (17,16) (19,18)
slot 13 (0,17) (2,6) (3,4) (5,16) (7,10) (8,14) (9,1) (12,19) (13,15) (18,11)
slot 14 (0,14) (1,10) (6,18) (7,8) (11,9) (13,3) (15,12) (16,2) (17,5) (19,4)
slot 15 (2,1) (3,17) (4,7) (6,12) (9,19) (10,0) (11,8) (14,13) (16,15) (18,5)
slot 16 (1,2) (3,6) (4,14) (5,18) (7,11) (8,17) (12,16) (13,9) (15,10) (19,0)
slot 17 (0,7) (2,9) (5,12) (8,3) (11,16) (13,4) (14,10) (15,19) (17,1) (18,6)
slot 18 (0,8) (1,15) (2,14) (6,7) (9,5) (10,11) (12,17) (16,3) (18,4) (19,13)
slot 19 (1,18) (3,19) (4,9) (7,2) (8,13) (10,6) (11,0) (12,14) (16,5) (17,15)
slot 20 (2,12) (4,3) (5,9) (6,17) (8,0) (13,7) (14,11) (15,1) (18,10) (19,16)
slot 21 (0,3) (1,19) (2,17) (6,13) (7,12) (9,8) (10,5) (11,4) (14,15) (16,18)
slot 22 (3,11) (4,10) (5,6) (12,0) (13,14) (15,2) (16,1) (17,8) (18,9) (19,7)
slot 23 (0,15) (1,16) (3,5) (8,6) (9,11) (10,7) (14,4) (17,12) (18,13) (19,2)
slot 24 (0,19) (1,17) (2,4) (5,13) (6,15) (7,14) (8,16) (9,3) (11,18) (12,10)

slot 25 (3,9) (4,8) (6,0) (10,2) (13,1) (14,18) (15,5) (16,11) (17,7) (19,12)
slot 26 (2,0) (4,13) (7,5) (9,17) (10,3) (12,6) (14,8) (15,16) (18,1) (19,11)
slot 27 (0,4) (1,3) (2,18) (5,10) (7,15) (8,9) (11,12) (13,19) (16,6) (17,14)
slot 28 (3,1) (5,4) (6,11) (8,12) (9,7) (10,15) (14,2) (16,19) (17,13) (18,0)
slot 29 (2,8) (4,17) (6,1) (7,0) (11,10) (12,5) (13,16) (14,9) (15,18) (19,3)
slot 30 (0,6) (1,5) (3,14) (8,11) (9,2) (10,16) (12,13) (15,4) (17,19) (18,7)
slot 31 (0,2) (3,12) (5,8) (6,4) (7,17) (9,18) (10,1) (13,11) (16,14) (19,15)
slot 32 (1,9) (2,5) (4,16) (6,19) (11,7) (12,15) (13,10) (14,0) (17,3) (18,8)
slot 33 (0,18) (4,1) (5,3) (7,6) (8,2) (9,13) (14,17) (15,11) (16,12) (19,10)
slot 34 (1,7) (2,16) (3,0) (6,9) (10,4) (11,5) (12,8) (15,13) (18,17) (19,14)
slot 35 (1,12) (4,2) (5,19) (6,3) (7,16) (8,15) (9,10) (11,14) (13,0) (17,18)
slot 36 (0,10) (2,19) (3,18) (5,7) (12,11) (13,8) (14,1) (15,9) (16,4) (17,6)
slot 37 (0,9) (3,15) (4,6) (7,13) (8,5) (10,14) (11,2) (16,17) (18,12) (19,1)

Πρόβλημα Late 15

Βρήκαμε την λύση (0, 1280) που έχει ανάλυση κόστους

CA3: 10
BR1: 10
BR2: 1260

και ωρολόγιο πρόγραμμα:

slot 0 (0,19) (1,8) (5,2) (6,15) (7,4) (9,3) (10,18) (12,14) (13,16) (17,11)
slot 1 (2,1) (3,5) (4,9) (11,0) (12,8) (14,7) (15,13) (16,17) (18,6) (19,10)
slot 2 (0,16) (1,5) (3,2) (6,4) (7,11) (9,12) (10,8) (13,14) (17,19) (18,15)
slot 3 (2,19) (4,3) (5,18) (7,10) (8,9) (11,13) (12,6) (15,14) (16,1) (17,0)
slot 4 (0,18) (1,15) (3,7) (6,17) (9,11) (10,4) (13,12) (14,5) (16,2) (19,8)
slot 5 (1,9) (2,18) (3,14) (4,16) (5,13) (7,12) (8,0) (10,6) (11,19) (15,17)
slot 6 (0,1) (6,2) (8,11) (9,5) (12,15) (13,7) (14,10) (17,4) (18,3) (19,16)
slot 7 (1,6) (2,12) (3,10) (4,8) (5,17) (7,0) (13,9) (15,19) (16,14) (18,11)
slot 8 (0,6) (4,1) (7,5) (8,3) (9,16) (10,15) (11,12) (14,2) (17,13) (19,18)
slot 9 (2,0) (5,19) (6,16) (8,7) (12,3) (13,1) (14,11) (15,9) (17,10) (18,4)
slot 10 (0,5) (3,13) (4,15) (7,2) (9,17) (10,1) (11,6) (12,18) (16,8) (19,14)
slot 11 (1,7) (2,13) (5,10) (6,3) (9,19) (12,0) (14,4) (15,8) (16,11) (18,17)
slot 12 (0,15) (4,5) (7,6) (8,14) (10,9) (11,3) (13,18) (16,12) (17,2) (19,1)
slot 13 (1,17) (2,8) (3,19) (6,9) (10,11) (12,5) (13,4) (14,0) (15,16) (18,7)
slot 14 (0,13) (1,3) (2,10) (4,12) (5,6) (9,14) (11,15) (16,18) (17,8) (19,7)
slot 15 (3,17) (5,16) (7,9) (8,13) (10,0) (11,4) (12,19) (14,6) (15,2) (18,1)

slot 16 (0,9) (1,12) (2,4) (5,11) (6,8) (15,3) (16,10) (17,7) (18,14) (19,13)
slot 17 (3,16) (4,0) (6,19) (7,15) (8,5) (9,18) (11,2) (12,17) (13,10) (14,1)
slot 18 (0,3) (1,11) (2,9) (10,12) (13,6) (14,17) (15,5) (16,7) (18,8) (19,4)
slot 19 (2,15) (3,1) (4,14) (5,0) (6,13) (8,10) (9,7) (11,18) (17,16) (19,12)
slot 20 (0,2) (1,14) (4,6) (7,8) (10,19) (12,9) (13,11) (16,15) (17,3) (18,5)
slot 21 (2,16) (5,14) (7,3) (8,6) (9,4) (11,17) (12,1) (13,0) (15,10) (18,19)
slot 22 (0,17) (1,2) (3,15) (4,18) (6,5) (8,16) (9,13) (10,7) (14,12) (19,11)
slot 23 (2,14) (3,8) (5,1) (7,17) (11,9) (12,13) (15,4) (16,6) (18,10) (19,0)
slot 24 (0,7) (1,19) (4,11) (6,12) (8,2) (10,5) (13,3) (14,16) (15,18) (17,9)
slot 25 (1,0) (2,3) (5,8) (7,18) (9,6) (11,10) (12,4) (14,15) (16,13) (19,17)
slot 26 (3,0) (5,7) (6,1) (8,19) (10,14) (13,2) (15,11) (16,4) (17,12) (18,9)
slot 27 (0,12) (2,6) (4,10) (7,1) (9,15) (11,16) (14,8) (17,5) (18,13) (19,3)
slot 28 (3,18) (4,2) (5,9) (8,1) (10,17) (11,14) (12,7) (13,19) (15,6) (16,0)
slot 29 (0,8) (1,10) (3,4) (6,11) (7,13) (9,2) (12,16) (14,18) (17,15) (19,5)
slot 30 (2,7) (3,12) (4,17) (6,14) (8,18) (9,10) (11,1) (13,5) (15,0) (16,19)
slot 31 (1,13) (5,4) (7,16) (10,3) (11,8) (12,2) (14,9) (17,6) (18,0) (19,15)
slot 32 (0,14) (2,11) (3,6) (4,19) (8,12) (9,1) (10,13) (15,7) (16,5) (17,18)
slot 33 (1,16) (5,15) (6,0) (8,4) (11,7) (12,10) (13,17) (14,3) (18,2) (19,9)
slot 34 (0,11) (4,13) (5,3) (6,18) (7,14) (9,8) (10,16) (15,12) (17,1) (19,2)
slot 35 (0,10) (1,4) (2,5) (3,9) (6,7) (8,17) (12,11) (13,15) (14,19) (18,16)
slot 36 (4,7) (9,0) (10,2) (11,5) (13,8) (15,1) (16,3) (17,14) (18,12) (19,6)
slot 37 (0,4) (1,18) (2,17) (3,11) (5,12) (6,10) (7,19) (8,15) (14,13) (16,9)

□

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.