UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF ELECTRONICS AND COMPUTERS
LABORATORY OF WIRE COMMUNICATIONS AND INFORMATION TECHNOLOGY

AN AI APPOACH TO LARGE SCALE MEDICAL APPOINTMENT (RE)SCHEDULING USING ANSWER SET PROGRAMMING

D I P L O M A T H E S I S

STAVROS KANIAS

SUPERVISOR: PAVLOS PEPPAS

PATRAS – OCTOBER 2023

University of Patras, Department of Electrical and Computer Engineering.

Stavros Kanias

CERTIFICATION

It is certified that the Diploma Thesis titled

AN AI APPOACH TO LARGE SCALE MEDICAL APPOINTMENT (RE)SCHEDULING USING ANSWER SET PROGRAMMING

of the Department of Electrical and Computer Engineering student

STAVROS KANIAS

Registration Number: 1066563

was presented publicly at the Department of Electrical and Computer Engineering at

……../……../……

and was examined by the following examining committee:

Name Surname, Title, Affiliation (supervisor)

Name Surname, Title, Affiliation (committee member)

Name Surname, Title, Affiliation (committee member)

The Supervisor	The Director of the Division
Name Surname Title	Name Surname Title

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΕΝΣΥΡΜΑΤΗΣ ΕΠΙΚΟΙΝΩΝΙΝΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΕΞΥΠΝΟ ΣΥΣΤΗΜΑ (ΕΠΑΝΑ)ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΥΓΕΙΟΝΟΜΙΚΩΝ ΡΑΝΤΕΒΟΥ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΣΥΝΟΛΟΥ ΑΠΑΝΤΗΣΕΩΝ

Δ Ι Π Λ Ω Μ Α Τ Ι Κ Η Ε Ρ Γ Α Σ Ι Α

ΣΤΑΥΡΟΣ ΚΑΝΙΑΣ

ΕΠΙΒΛΕΠΩΝ: ΠΑΥΛΟΣ ΠΕΠΠΑΣ

ΠΑΤΡΑ – ΟΚΤΩΒΡΙΟΣ 2023

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών.

Σταύρος Κάνιας

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

ΕΞΥΠΝΟ ΣΥΣΤΗΜΑ (ΕΠΑΝΑ)ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΥΓΕΙΟΝΟΜΙΚΩΝ ΡΑΝΤΕΒΟΥ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΣΥΝΟΛ ΑΠΑΝΤΗΣΕΩΝ

του/της φοιτητή/τριας του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΣΤΑΥΡΟΣ ΚΑΝΙΑΣ ΤΟΥ **ΝΕΚΤΑΡΙΟΥ**

Αριθμός Μητρώου: 1066563

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

……../……../……

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (επιβλέπων)

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (μέλος επιτροπής)

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (μέλος επιτροπής)

Ο/Η Επιβλέπων/ουσα	Ο/Η Διευθυντής/τρια του Τομέα
Ονοματεπώνυμο Βαθμίδα	Ονοματεπώνυμο Βαθμίδα

PREFACE

Acknowledgements

The Preface is optional.

Publications

ABSTRACT

AN AI APPOACH TO LARGE SCALE MEDICAL APPOINTMENT (RE)SCHEDULING USING ANSWER SET PROGRAMMING

Student: Stavros Kanias **Supervisor: Pavlos Peppas**

Despite their significant solving capabilities ASP solvers have not yet crossed the line between academic and widespread industrial use. Even in areas where classical AI solutions developed in the Answer Set Programing (ASP) model exceed solvers that utilize procedural algorithms in terms of raw performance, the latter approach is usually preferred. The lack of integration between ASP solvers and standard software development components, such as SQL databases, is commonly cited as a reason for this tendency. Furthermore, although declarative programing has proved itself as a capable programing model, its dependency on a fact containing knowledge base to describe the initial state of reality creates a challenge in the case where this reality is characterized by high complexity due to its many parameters and the scale of data produced by the modern software applications.

To this end, it is of outmost importance for the Knowledge Representation community to focus on making ASP solvers compatible with the rest of the software industry infrastructure and creating tools to automatically convert large scale data into an ASP knowledge base using encodings that preserve the expressiveness of the declarative model. This will enable ASP programs to keep resembling natural language whilst simultaneously possessing the ability to seamlessly handle large volumes of data. In this case a strong incentive will be given to any software engineer to adopt and integrate ASP into their projects as the plethora of its benefits most notably the code readability, the optimization efficiency and the reduced development time will come with an insignificant migration cost.

With all the above in mind, during the course of this Diploma Thesis a Python interface for converting an SQL database to a logic program knowledge base was developed. This knowledge base is dynamically provided as input to any desired ASP clingo program able to decode and use its data according to the program's predicate and rule vocabulary. This interface supports all the basic SQL transactions adding a level of abstraction on top of the Potassco backed CLORM module thus greatly simplifying the data flow between a software application's backend and an ASP solver, in this case the clingo solver. The developed interface was applied in the implementation of a new, AI driven, approach to medical appointment (re)scheduling which for the first time possessed the foresight to maximize the total community benefit instead of the individual one. In this way the immense potential of ASP in the modern software industry is showcased. In summary, this Diploma Thesis provides an in-depth description of the final application produced by the utilization of the ASP-SQL interface to solve the medical appointment (re)scheduling problem.

ΕΚΤΕΤΑΜΕΝΗ ΕΛΛΗΝΙΚΗ ΠΕΡΙΛΗΨΗ

ΤΙΤΛΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Φοιτητής: Σταύρος Κάνιας **Επιβλέπων: Παύλος Πέππας**

1. Εισαγωγή

Από την εμφάνισή του το 1972 [1] μέχρι και σήμερα ο λογικός προγραμματισμός έχει εμφανίσει αλματώδη ανάπτυξη σε βαθμό όπου οι λογικοί επιλυτές, όπως η clingo, είναι ικανοί να διαχειριστούν ιδιαίτερα σύνθετα, άρτια και ακαθόριστης πολυωνυμικής πολυπλοκότητας (NP-complete) προβλήματα βελτιστοποίησης εφόσον έχει προηγηθεί η έκφρασή τους υπό τη σημασιολογία του Answer Set Programming (ASP)[1]. Αρκετές προσπάθειες έχουν ήδη πραγματοποιηθεί ώστε τα πολλαπλά οφέλη του ΠΣΑ, κυρίως η υψηλή αναγνωσιμότητα του δηλωτικού κώδικα λόγω της ομοιότητάς του με τη φυσική γλώσσα και ο μειωμένος χρόνος ανάπτυξης εφαρμογών λόγω της προσπέρασης του σταδίου ανάπτυξης αλγορίθμου βελτιστοποίησης, να αξιοποιηθούν από τους κλάδους της Τεχνητής Νοημοσύνης, της Βιοπληροφορικής και των Βάσεων Δεδομένων κάνοντας την υιοθέτησή της τεχνολογίας αυτής από την βιομηχανία λογισμικού θέμα χρόνου [2]. Ειδικά σε έργα μεγάλης κλίμακας με ανάγκες πολυπαραγοντικής βελτιστοποίησης όπως η ψηφιοποίηση ενός υγειονομικού συστήματος με ταυτόχρονες απαιτήσεις υψηλής αξιοπιστίας, καθώς διακυβεύεται η ανθρώπινη υγεία αλλά και ευχρηστίας, καθώς είναι θεμιτή η διευκόλυνση της αλληλεπίδρασης του ασθενή με το σύστημα ώστε οι παρεχόμενες υπηρεσίες να είναι εύκολα προσβάσιμες σε όλους, η υιοθέτηση τεχνολογιών Τεχνητής Νοημοσύνης είναι μονόδρομος. Ένα από τα σημεία που καθημερινά αναδεικνύουν την αδυναμία των σύγχρονων συστημάτων υγείας να διαχειριστούν το φόρτο που τους ανατίθεται με βέλτιστο τρόπο, κυρίως σε χώρες όπου η ζήτηση ξεπερνά κατά πολύ τις δυνατότητες παροχής υπηρεσιών από το ιατρικό προσωπικό, είναι η διαδικασία διαχείρισης των ακυρώσεων στα εξωτερικά ραντεβού. Συγκεκριμένα, η διαχείριση των ακυρώσεων με πολιτική που θέτει ως προτεραιότητα το κοινωνικό όφελος κατά τη βελτιστοποίηση εγείρει προκλήσεις λόγω του τεράστιου όγκου δεδομένων που εμφανίζεται στα υγειονομικά συστήματα αλλά και της ζωτικής σημασίας κάθε απόφασης που λαμβάνεται από αυτό αφού διακύβευμα αποτελεί η ανθρώπινη ζωή.

Κινούμενη στο παραπάνω πλαίσιο η παρούσα διπλωματική εργασία εξετάζει την επιστράτευση της τεχνητής νοημοσύνης και συγκεκριμένα του ΠΣΑ για την υλοποίηση καινοτόμων μεθόδων διαχείρισης των ακυρώσεων σε σύγχρονα ψηφιακά υγειονομικά συστήματα. Για την ανάπτυξη όμως μια τέτοιας εφαρμογής κρίθηκε απαραίτητη η ανάπτυξη μιας διεπαφής μεταξύ κάποιας γλώσσας προγραμματισμού υψηλού επιπέδου, στην συγκεκριμένη περίπτωση της Python, και ενός λογικού επιλυτή, συγκεκριμένα της clingo για τη διευκόλυνση της ροής δεδομένων κατά την παροχή δεδομένων από τη βάση δεδομένων στον επιλυτή και κατά την ενημέρωση της βάσης δεδομένων με τα δεδομένα που προκύπτουν από την έξοδο του επιλυτή. Από τα πρώτα ακόμα στάδια εκπόνησης της εργασίας έγινε εμφανές το σημαντικό κενό μεταξύ του ΠΣΑ και της βιομηχανίας λογισμικού, φαινόμενο που συχνά αποδίδεται στην έλλειψη διεπαφών μεταξύ των δομικών στοιχείων που απαρτίζουν τις σύγχρονες εφαρμογές λογισμικού, όπως οι βάσεις δεδομένων, και των λογικών επιλυτών. Παραδείγματος χάρη, παρότι έχουν υπάρξει προσπάθειες χρήσης του ΠΣΑ στον κλάδο των βάσεων δεδομένων και συγκεκριμένα στη βελτιστοποίηση της λειτουργίας των συστημάτων διαχείρισης βάσεων δεδομένων (DBMS) [3] μέχρι στιγμής δεν

[1] Ως ελληνικός όρος χρησιμοποιείται ο *προγραμματισμός συνόλων απαντήσεων* (ΠΣΑ)

υπάρχει ανοιχτά διαθέσιμος κάποιος αυτοματοποιημένος τρόπος μετατροπής μιας βάσης δεδομένων σε βάση γνώσης ενός λογικού προγράμματος εντός κάποιου προγραμματιστικού περιβάλλοντος υψηλού επιπέδου το οποίο θα επέλεγε ρεαλιστικά ένας σύγχρονος μηχανικός λογισμικού για να υλοποιήσει κάποιο έργο. Ταυτόχρονα αισθητή είναι η απουσία κάποιου συστήματος διαχείρισης βάσης γνώσης (KBMS) αντίστοιχο αυτών των βάσεων δεδομένων (DMBS).

Το τελικό αποτέλεσμα αυτής της εργασίας είναι μια πλήρως λειτουργική εφαρμογή επίλυσης του προβλήματος (επανα)προγραμματισμού υγειονομικών ραντεβού για ένα υγειονομικό σύστημα που χρησιμοποιεί βάση δεδομένων τεχνολογίας PostgreSQL, ως επιλυτή την clingo και με τη γλώσσα Python να λειτουργεί ως δίαυλο επικοινωνίας μεταξύ των δύο. Η εφαρμογή αυτή προορίζεται για χρήση εντός του τμήματος διαχείρισης δεδομένων (backend) μιας διαδικτυακής υγειονομικής πλατφόρμας με την οποία θα αλληλεπιδρούν άμεσα οι πολίτες για την εξυπηρέτησή τους από το σύστημα. Η γλώσσα προγραμματισμού Python χρησιμοποιείται για την υλοποίηση των διεπαφών μεταξύ βάσης δεδομένων και του λογικού επιλυτή μετατρέποντας τη βάση δεδομένων σε βάση γνώσης, παρέχοντας παράλληλα δυνατότητες εκτέλεσης ερωτημάτων και γενικότερα SQL λειτουργιών εντός της βάσης γνώσης. Στην ενότητα 2 παρουσιάζεται το πρόβλημα του επαναπρογραμματισμού υγειονομικών ραντεβού μετά από ακύρωση, το πραγματικό του κόστος και οι μέχρι τώρα εφαρμοζόμενες μέθοδοι διαχείρισής του. Έπειτα, προτείνεται για πρώτη φορά η μέθοδος βελτιστοποίησης κοινού οφέλους βασιζόμενη στην τεχνητή νοημοσύνη, η οποία αποτελεί και τη βασική συνεισφορά αυτή της εργασίας. Στην ενότητα 3 αναλύεται λεπτομερώς η εφαρμογή που αναπτύχθηκε για την επίλυση του προβλήματος. Στην πρώτη υποενότητα αναλύεται ο σχεδιασμός της βάσης δεδομένων μέχρι την ανάπτυξη του λογικού κώδικα για τον επιλυτή. Έπειτα παρουσιάζεται η δομή των δύο διεπαφών, η πρώτη μεταξύ Python και PostgreSQL και η δεύτερη μεταξύ Python και Clingo. Τέλος, γίνεται εκτενής αναφορά στο σχεδιασμό του λογικού κώδικα και των εννοιών που αυτός οφείλει να διασφαλίζει σε ένα υγειονομικό σύστημα. Ακολουθεί ενότητα αξιολόγησης της εφαρμογής βασισμένη σε ψευδορεαλιστικά υγειονομικά συστήματα που δημιουργήθηκαν μέσω της Python χρησιμοποιώντας την αρχιτεκτονική της βάσης δεδομένων. Τα αποτελέσματα είναι αρκετά υποσχόμενα. Η πέμπτη και τελευταία ενότητα πραγματεύεται ικανότητα της εφαρμογής να χρησιμοποιηθεί στον πραγματικό κόσμο, τις δοκιμασίες που θα συναντήσει και τα μέτρα που μπορούν να ληφθούν για την επίλυσή τους.

Το πρόβλημα του επαναπρογραμματισμού υγειονομικών ραντεβού

Έστω ένα νοσοκομείο που εξυπηρετεί αρκετές χιλιάδες ασθενείς σε εβδομαδιαία βάση και απασχολεί δεκάδες γιατρούς. Ας υποθέσουμε επίσης ότι κάθε γιατρός εργαζόμενος 8 ώρες την ημέρα προσφέρει στο σύστημα 8 ραντεβού της μιας ώρας από Δευτέρα μέχρι Παρασκευή. Στα περισσότερα σύγχρονα υγειονομικά συστήματα η διαδικασία εξασφάλισης ενός ραντεβού από έναν ασθενή έχει ως εξής. Αρχικά ο ασθενής είτε βλέπει το παρόν πρόγραμμα σε μια πλατφόρμα είτε καλεί στο κέντρο επικοινωνίας του υγειονομικού συστήματος. Του παραχωρείται στη συνέχεια ένα διαθέσιμο ραντεβού το οποίο κατά πάσα πιθανότητα θα εμφανίζεται μετά από ένα αρκετά μεγάλο χρονικό διάστημα. Σε περίπτωση ακύρωσης το ραντεβού αδειάζει και εάν η ακύρωση γίνει χρονικά κοντά στην ημερομηνία του ραντεβού το πιθανότερο είναι αυτό το ραντεβού να μην αξιοποιηθεί από κανέναν ασθενή. Αυτό το σύστημα, παρότι δεν είναι φανερό εκ πρώτης όψεως έχει σημαντικό και πολυεπίπεδο κόστος κυρίως για τα χαμηλότερα κοινωνικά στρώματα. Ας αναλογιστούμε ότι ο μέσος όρος αναμονής για ένα ιατρικό ραντεβού στις ΗΠΑ το 2022 ήταν 26 μέρες [4] αναγκάζοντας 41% των ενηλίκων να λάβουν καθυστερημένη θεραπεία για τη νόσο τους εκ των οποίων το 12% διαγνώστηκε με νόσο που απαιτούσε άμεση θεραπεία [5]. Ταυτόχρονα

δεν μπορούμε να αγνοήσουμε το γεγονός ότι από το ποσοστό των ασθενών που έλαβαν θεραπεία με καθυστέρηση το 51.7% είχε ετήσιο εισόδημα χαμηλότερο των 50.000 δολαρίων, εισόδημα ιδιαίτερα χαμηλό για τις ΗΠΑ αυτή τη χρονική περίοδο. Τέλος, το κόστος σε κεφάλαιο με το οποίο τα χαμένα ιατρικά ραντεβού επιβαρύνουν το σύστημα υγείας των ΗΠΑ ανέρχεται στα 150 δισεκατομμύρια δολάρια [6].

Τα παραπάνω δεδομένα είναι αρκετά για να συμπεράνουμε ότι συστήματα υγείας που χρησιμοποιούν τη συμβατική μέθοδο διαχείρισης ραντεβού πάσχουν και βρίσκονται σε άμεση ανάγκη εύρεσης κάποιας αποδοτικότερης προσέγγισης επαναπρογραμματισμού των ακυρωμένων υγειονομικών ραντεβού. Μια πρώτη σκέψη που πιλοτικά έχει δοκιμαστεί είναι η δημιουργία σειρών αναμονής για τα ραντεβού ώστε στην περίπτωση ακύρωσης το ραντεβού να περνά στον πρώτο επιλαχών. Αν και καλύτερη από τη συμβατική, η προσέγγιση αυτή, όπως θα αναδειχθεί παρακάτω, δεν είναι βέλτιστη εάν η παράμετρος που προσπαθεί να βελτιστοποιήσει το σύστημα είναι το κοινό όφελος. Είναι προφανές πως σε μια προσέγγιση εξυπηρέτησης του πρώτου επιλαχόντος μεγιστοποιείται μεν το ατομικό όφελος αλλά ταυτόχρονα παραβλέπονται επαναπρογραμματισμοί που υπό συνθήκες θα μεγιστοποιούσαν την προσφορά του συστήματος στο κοινό σύνολο χωρίς να ζημιώνουν στην πράξη το άτομο.

Πριν προχωρήσουμε στην προτεινόμενη από αυτή την εργασία προσέγγιση είναι επίσης απαραίτητο να διασαφηνίσουμε τις απαιτήσεις που θα πρέπει να ικανοποιεί ένα ρεαλιστικό σύστημα επαναπρογραμματισμού ιατρικών ραντεβού τόσο σε εννοιολογικό αλλά και σε πρακτικό επίπεδο. Σε επίπεδο σχεδιασμού το σύστημα θα πρέπει να διασφαλίζει τις εξής συνθήκες:

1. Κάθε ασθενής που **έχει ραντεβού** στο **παρόν** πρόγραμμα θα πρέπει να **έχει ραντεβού** και στο **νέο** πρόγραμμα που θα προκύψει από τον επαναπρογραμματισμό.
2. Κανένας ασθενής δεν θα πρέπει ως αποτέλεσμα του επαναπρογραμματισμού να λάβει ραντεβού με λιγότερο όφελος από αυτό που ήδη κατέχει καθώς αυτή η ενέργεια θα θεωρηθεί ως αδικία απέναντι στο άτομο. Ως αποτέλεσμα το σύστημα έχει τη δυνατότητα να αναθέτει μόνο ραντεβού με μεγαλύτερο όφελος σε έναν ασθενή σε σχέση με αυτό που ήδη έχει στην κατοχή του. Έχοντας λάβει την προτεραιότητα ως σταθερά από σημαίνει ότι ο επαναπρογραμματισμός θα οδηγεί μόνο σε αναθέσεις ραντεβού **μεγαλύτερης προτίμησης** για κάθε ασθενή εξασφαλίζοντας ότι δεν θα ζημιωθεί στην πράξη το ατομικό όφελος.

Ως προς την υλοποίηση το σύστημα για να μπορεί να ανεπεξέλθει στην πραγματικότητα θα πρέπει να εμφανίζει τα παρακάτω χαρακτηριστικά:

1. Υψηλές υπολογιστικές ικανότητες ώστε να μπορέσει να διαχειριστεί το τεράστιο όγκο δεδομένων που εμφανίζεται στα υγειονομικά συστήματα.
2. Τεχνολογική συμβατότητα με τα ήδη υπάρχοντα συστήματα λογισμικού που χρησιμοποιούνται στον τομέα της υγείας.
3. Δικαιοσύνη απέναντι στους ασθενείς αφού το διακύβευμα είναι η υγεία και η ανθρώπινη ζωή.
4. Ευχρηστία και ευχάριστη εμπειρία χρήσης ώστε να μπορεί να χρησιμοποιηθεί από τη μεγαλύτερη δυνατή μερίδα της κοινωνίας και να παρέχει στους χρήστες το κίνητρο να συνεχίσουν να χρησιμοποιούν τη διαδικτυακή πλατφόρμα διαχείρισης ραντεβού.
5. Δυνατότητα κλιμάκωσης ανάλογα με το μέγεθος του οργανισμού που χρησιμοποιεί το σύστημα.
6. Γλωσσική ευχέρεια για την περιγραφή σύνθετων καταστάσεων όπως ηθικά διλλήματα τα οποία συχνά εμφανίζονται στον τομέα της υγείας.
7. Έξοδο ικανή να περιγράψει τη σειρά των ενεργειών επαναπρογραμματισμού ώστε να μεταβεί το σύστημα από το αρχικό στο νέο πρόγραμμα.

Για να γίνει κατανοητή η προσέγγιση που προτείνεται από αυτή την εργασία είναι απαραίτητος ο ποσοτικός ορισμός του κοινού οφέλους. Για τις ανάγκες της εργασίας αυτής, ως κοινό όφελος ορίζουμε το άθροισμα του οφέλους (score) κάθε επιβεβαιωμένου ραντεβού στο σύστημα. Το όφελος του κάθε ραντεβού με τη σειρά του δίνεται από τον τύπο (1).

$$Όφελος = \frac{0.3}{προτίμηση} + 0.7 \cdot προτεραιότητα \quad (1)$$

Ως προτίμηση ορίζουμε τη σειρά με την οποία ο ασθενής έχει επιλέξει το συγκεκριμένο ραντεβού και επιδρά αντιστρόφως ανάλογα στην τιμή του οφέλους. Για παράδειγμα η πρώτη επιλογή ενός ασθενή θα έχει τιμή προτίμησης ίση με 1 μεγιστοποιώντας το όφελος. Η προτεραιότητα είναι μια παράμετρος που εξάγεται από το ιατρικό ιστορικό του ασθενή σταθμίζοντας τις παραμέτρους που καθιστούν την κατάστασή του περισσότερο ή λιγότερο επείγουσα. Αξίζει να τονιστεί ότι ο τρόπος αυτός για τον υπολογισμό του οφέλους είναι κατά βάση αυθαίρετος παρότι αρκετά λογικός για να αναπτυχθεί μια εφαρμογή βασισμένη σε αυτόν ως μια πρώτη υλοποίηση. Κάλλιστα μπορεί σε μια μελλοντική υλοποίηση να εισαχθούν νέες παράμετροι και να αλλάξουν τα βάρη που έχουν οριστεί για τις δύο παραπάνω παραμέτρους.

Λαμβάνοντας τα παραπάνω ως αξιωματικές έννοιες στο σχεδιασμό του συστήματος καθίσταται φανερό πως το πρόβλημα αποτελεί γόνιμο έδαφος για την εφαρμογή του ΠΣΑ. Στο παράδειγμα που ακολουθεί περιγράφεται ένα πρόγραμμα όπου έξι ασθενείς διεκδικούν τέσσερα ραντεβού. Στην αρχική κατάσταση ο Νίκος έχει στην κατοχή του το ραντεβού της Δευτέρας με τη διατεταγμένη σειρά αναμονής να έχει κατά σειρά επιλαχόντες το Γιώργο και τον Κώστα. Η απόφαση του Νίκου να ακυρώσει το ραντεβού του θα προκαλέσει επαναπρογραμματισμό ο οποίος, εάν επιλεγεί η πολιτικής επιλογής του πρώτου επιλαχόντος, θα οδηγήσει στην ανάθεση του ραντεβού στο Γιώργο αφού το ραντεβού πρόκειται για μια καλύτερη επιλογή του. Το πρόγραμμα στην αρχική κατάσταση φαίνεται στον Πίνακα 1 ενώ το πρόγραμμα μετά από επαναπρογραμματισμό επιλογής πρώτου επιλαχόντος φαίνεται στο Πίνακα 2. Όπως προκύπτει από τον Πίνακα 2, στην περίπτωση αυτή το υγειονομικό σύστημα θα παρέχει κοινό όφελος ίσο με 258.

Δευτέρα	Ασθενής - Όφελος
9-10	Νίκος – 90
	Γιώργος – 87
	Κώστας – 86
Τρίτη	**Ασθενής – Όφελος**
13-14	Γιώργος - 72
	Γιάννης - 60
	Μαρία - 55
Πέμπτη	**Ασθενής - Όφελος**
11-12	Γιάννης - 45
	Μαρία - 40
Παρασκευή	**Ασθενής - Όφελος**
15-16	Κώστας - 71
	Δέσποινα - 70

Πίνακας 1: Αρχικό πρόγραμμα.

Δευτέρα	Ασθενής - Όφελος
9-10	Γιώργος – 87
	Κώστας – 86
Τρίτη	**Ασθενής – Όφελος**
13-14	Γιάννης - 60
	Μαρία - 55
Πέμπτη	**Ασθενής - Όφελος**
11-12	Μαρία - 40
Παρασκευή	**Ασθενής - Όφελος**
15-16	Κώστας - 71
	Δέσποινα - 70

Πίνακας 2: Νέο πρόγραμμα μετά από επαναπρογραμματισμό μεγιστοποίησης ατομικού οφέλους.

Δευτέρα	Ασθενής - Όφελος
9-10	Κώστας – 86
	Γιώργος – 87
Τρίτη	**Ασθενής - Όφελος**
13-14	Γιώργος - 72
	Γιάννης - 60
	Μαρία - 55
Πέμπτη	**Ασθενής - Όφελος**
11-12	Γιάννης - 45
	Μαρία - 40
Παρασκευή	**Ασθενής – Όφελος**
15-16	Δέσποινα - 70

Πίνακας 1: Νέο πρόγραμμα μετά από επαναπρογραμματισμό μεγιστοποίησης κοινού οφέλους.

Με βάση τα αξιώματα που αναφέρθηκαν παραπάνω η βασική πρόταση της νέας μεθόδου που προτείνεται σε αυτή την εργασία περικλείεται στην παρακάτω πρόταση.

Είναι αποδεκτή η μη επιλογή του πρώτου επιλαχόντος για την ανάθεση ενός ακυρωμένου ραντεβού εφόσον ο ασθενής αυτός έχει ήδη στην κατοχή του ένα ραντεβού και η επιλογή αυτή οδηγεί μέσω μιας αλυσίδας ενεργειών σε ένα νέο πρόγραμμα με μεγαλύτερο κοινό όφελος.

Εφαρμόζοντας την παραπάνω πρόταση στο πρόγραμμα του Πίνακα 1 και προβλέποντας το μεγαλύτερο κοινό όφελος που η επιλογή αυτή θα επιφέρει, επιλέγουμε να μη δώσουμε το ραντεβού της Δευτέρας στο Γιώργο μετά την ακύρωση του Νίκου. Αντίθετα, το ραντεβού θα δοθεί στον Κώστα και ο Γιώργος θα κρατήσει το ραντεβού της Τρίτης. Με αυτή τη λογική κανένας ασθενής δεν αδικήθηκε άμεσα ενώ το νέο πρόγραμμα μετά τον επαναπρογραμματισμό, όπως φαίνεται στον Πίνακα 3, εμφανίζει κοινό όφελος ίσο με 273 αποτελώντας ταυτόχρονα και τη βέλτιστη λύση για τη συγκεκριμένη αρχική κατάσταση.

Εάν όμως συνειδητοποιήσουμε πως η πολυπλοκότητα του προβλήματος είναι εκθετική ως προς τον αριθμό των αιτημάτων από τους ασθενείς για το σύνολο των ραντεβού όπως φαίνεται στη Σχέση 2 ερχόμαστε αναπόφευκτα αντιμέτωποι με τα εξής ερωτήματα.

- Με ποιο τρόπο θα εντοπιστεί το βέλτιστο πρόγραμμα ως προς το κοινό όφελος σε ένα τόσο μεγάλο χώρο καταστάσεων;
- Ποια τεχνολογία θα μπορούσε να λάβει υπόψιν όλες τις παραμέτρους που θα διασφαλίζουν τα αξιώματα που τέθηκαν για τον επαναπρογραμματισμό ενώ ταυτόχρονα θα εκτελεί τη διαδικασία της βελτιστοποίησης σε ικανοποιητικό χρόνο ώστε να αποτελεί μια ρεαλιστική λύση για ένα μεγάλης κλίμακας υγειονομικό σύστημα;

Τα δύο παραπάνω ερωτήματα ήταν αυτά που οδήγησαν στην επιλογή του ΠΣΑ για την επίλυση του προβλήματος καθώς εκτός από τα πολλά επιπλέον οφέλη που προσφέρει σε μια τέτοιου είδους εφαρμογή, η προσέγγιση της τεχνητής νοημοσύνης και συγκεκριμένα της αναπαράστασης γνώσης σε αυτό το πρόβλημα παρέχει ταυτόχρονα μια ολιστική προσέγγιση στις ποικίλες παραμέτρους του αλλά και μια υπολογιστικά ικανή λύση να διαχειριστεί τη μεγάλη του πολυπλοκότητα. Πιο συγκεκριμένα η πολυπλοκότητα του προβλήματος δίνεται ως:

$$O(S^R) = O\left(S^{d \cdot T}\right) = O\left(S^{d \cdot D \cdot h \cdot t}\right) = O(2^{d \cdot D \cdot h \cdot t}) \quad (2)$$

Όπου S οι δυνατές καταστάσεις που μπορεί να λάβει ένα αίτημα για ραντεβού και R το πλήθος των αιτημάτων στο σύστημα. Εφόσον κάθε αίτημα μπορεί είτε να γίνει δεκτό είτε να παραμείνει στη σειρά αναμονής η κατάσταση ενός αιτήματος είναι στην πραγματικότητα μια δυαδική λογική μεταβλητή με τιμές αληθές (εγκρίθηκε) ή ψευδές (δεν εγκρίθηκε). Ως αποτέλεσμα η βάση της εκθετικής πολυπλοκότητας θα είναι το 2. Οι παράμετροι d και T χρησιμοποιούνται για να συνδέσουν τον αριθμό των αιτημάτων με την πραγματική μεταβλητή του προβλήματος που είναι το πλήθος των διαθέσιμων ραντεβού από το σύστημα. Η ζήτηση d παριστάνει τον αριθμό των αιτημάτων που επιτρέπει το σύστημα σε έναν ασθενή να πραγματοποιήσει ενώ οι ώρες επίσκεψης των γιατρών T αναφέρονται στο πλήθος των ραντεβού που μπορεί να παρέχει το σύστημα και προκύπτουν εάν πολλαπλασιάσουμε τον αριθμό των γιατρών D με τις ημερήσιες ώρες εργασίας τους h και το χρονικό διάστημα που καλύπτει το πρόγραμμα σε μέρες το οποίο συμβολίζουμε με t.

Υλοποίηση με χρήση PostgreSQL, Python και Clingo

Έχοντας προβεί σε ενδελεχή ανάλυση του προβλήματος οδηγούμαστε στο τμήμα περιγραφής της εφαρμογής λογισμικού που υλοποιεί το τον προτεινόμενη προσέγγιση για τον επαναπρογραμματισμό υγειονομικών ραντεβού. Το σύστημα αυτό αποτελείται από δύο βασικούς κόμβους ροής δεδομένων και μια διεπαφή που λειτουργεί ως δίαυλος επικοινωνίας

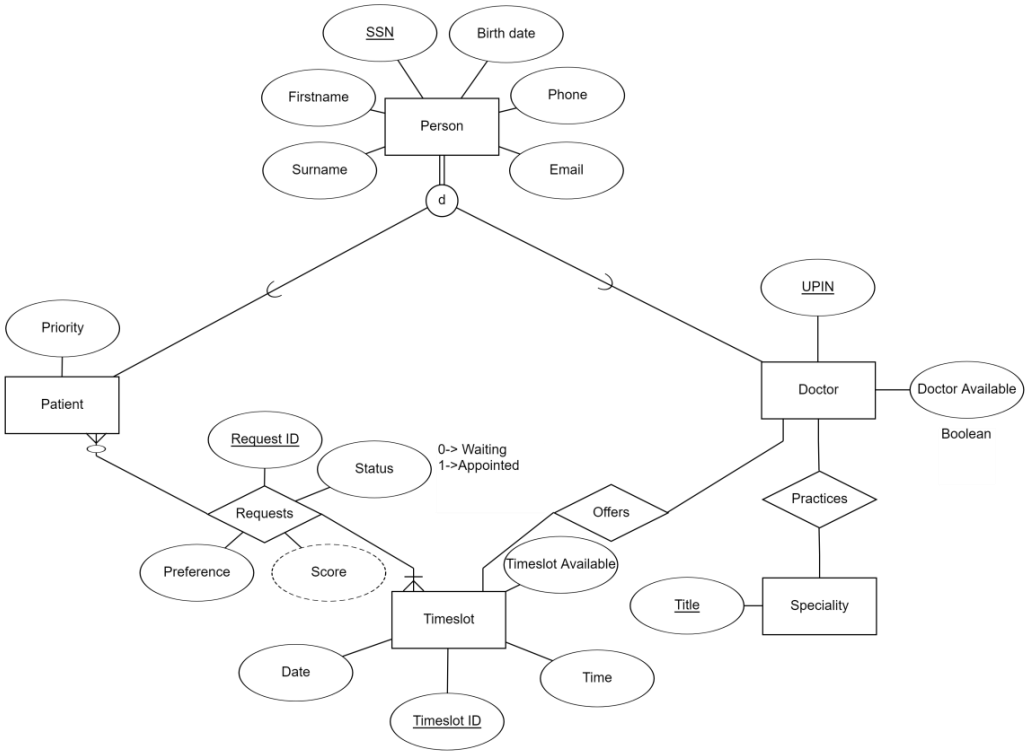
μεταξύ τους. Τα δεδομένα, όπως και σε κάθε σύγχρονη εφαρμογή λογισμικού αποθηκεύονται σε μια βάση δεδομένων και αντλούνται από αυτή κάθε φορά που το υπόλοιπο σύστημα τα ζητήσει. Η τεχνολογία που χρησιμοποιήθηκε για την υλοποίηση της βάσης δεδομένων, εφόσον η εφαρμογή αυτή προορίζεται για χρήση εντός μια διαδικτυακής πλατφόρμας, είναι η **PostgreSQL**. Η γλώσσα προγραμματισμού Python και συγκεκριμένα η βιβλιοθήκη **psycopg2** χρησιμοποιήθηκε για τη δημιουργία και τη διασφάλιση της σύνδεσης μεταξύ της βάσης δεδομένων και των υπόλοιπων στοιχείων του συστήματος. Στο δεύτερο κόμβο ροής δεδομένων του συστήματος βρίσκεται ο λογικός επιλυτής που στη συγκεκριμένη περίπτωση είναι ο επιλυτής **Clingo.** Στον κόμβο αυτό γίνεται όλη η επεξεργασία των δεδομένων ώστε να παραχθεί το νέο πρόγραμμα ραντεβού μετά από κάποια ακύρωση. Τα παραπάνω συνοψίζονται στο διάγραμμα που ακολουθεί.

Σχήμα 1: Διάγραμμα ροής δεδομένων για την προτεινόμενη εφαρμογή επαναπρογραμματισμού υγειονομικών ραντεβού.

Η βάση δεδομένων

Μια καλώς σχεδιασμένη και υλοποιημένη βάση δεδομένων είναι απαραίτητη για την ανάπτυξη και την αξιολόγηση κάθε σύγχρονης εφαρμογής. Στην παρούσα εργασία επιλέχθηκε το σχεσιακό μοντέλο για το σχεδιασμό της βάσης δεδομένων και η τεχνολογία SQL για την υλοποίησή της. Με τον τρόπο αυτό γίνεται πιο προφανής και εύκολη στην κατανόηση η δομή της βάσης δεδομένων στον αναγνώστη ενώ ταυτόχρονα διευκολύνεται και η μεταγραφή της βάσης δεδομένων σε βάση γνώσης λόγω της άμεσης λογική μετάβασης από την εγγραφή στο κατηγόρημα όπως θα φανεί και στην επόμενη υποενότητα. Η βάση δεδομένων της οποίας το διάγραμμα οντοτήτων-συσχετίσεων παρουσιάζεται στην Εικόνα 1 προσομοιώνει τις βασικές πτυχές ενός πραγματικού συστήματος υγείας δίνοντας τη δυνατότητα δοκιμής των υπολογιστικών δυνατοτήτων της εφαρμογής σε ρεαλιστικές συνθήκες. Το τελικό σχεσιακό σχήμα για την πλήρη περιγραφή της δομής της βάσης δεδομένων βρίσκεται στην Εικόνα 2. Για την ανάπτυξη του διαγράμματος οντοτήτων-συσχετίσεων χρησιμοποιήθηκε η εφαρμογή ERD Plus [7] ενώ για τη δημιουργία του σχεσιακού σχήματος η εφαρμογή DB designer [8].

Από τα αρχικά στάδια ανάπτυξης της εργασίας κατέστη φανερό πως και για την ανάπτυξη ψευδορεαλιστικών δεδομένων για τη δοκιμή του συστήματος αλλά και για τη δόμηση της βάσης γνώσης ήταν απαραίτητη μια μέθοδος κωδικοποίησης της βάσης δεδομένων με χρήση κάποιας δομής δεδομένων που παρέχεται από γλώσσες προγραμματισμού υψηλού επιπέδου όπως η Python. Η δομή που επιλέχθηκε τελικά είναι αυτή ενός πολυεπίπεδου λεξικού (multilevel hash map) με τα κλειδιά του πρώτου επιπέδου να αναπαριστούν τις οντότητες του σχεσιακού σχήματος. Τιμή κάθε κλειδιού πρώτου επιπέδου είναι ένα νέο λεξικό με κλειδιά που αναπαριστούν τα γνωρίσματα (attributes) της κάθε οντότητας και τιμή μια λίστα η οποία περιέχει την απαραίτητη πληροφορία για την αναπαράσταση της βάσης δεδομένων.

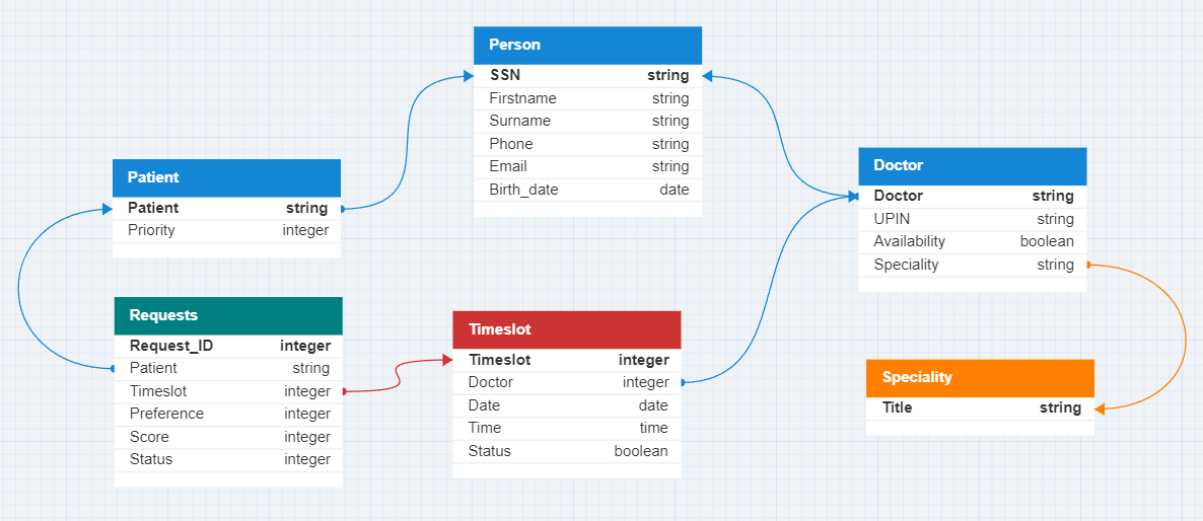


Εικόνα 1: Διάγραμμα οντοτήτων συσχετίσεων (ERD) για τη βάση δεδομένων.

Ανάλογα με το μέγεθος της λίστας αυτής και τα ιδιαίτερα χαρακτηριστικά του κάθε γνωρίσματος διακρίνουμε τρεις περιπτώσεις για το περιεχόμενο των τιμών ενός λεξικού δευτέρου επιπέδου.

1. Εάν η λίστα έχει δύο στοιχεία, το πρώτο αντιστοιχεί στον τύπο δεδομένων του γνωρίσματος και το δεύτερο σηματοδοτεί εάν το στοιχείο αυτό πρόκειται για πρωτεύον κλειδί.
2. Εάν η λίστα έχει τρία στοιχεία, τα πρώτα δύο έχουν την ίδια σημασία με τα δύο στοιχεία της περίπτωσης 1 και το τρίτο σηματοδοτεί εάν το γνώρισμα έχει μοναδική τιμή (unique)
3. Εάν η λίστα έχει τέσσερα στοιχεία τότε το γνώρισμα πρόκειται για ξένο κλειδί. Ως επακόλουθο τα δύο πρώτα στοιχεία της λίστας έχουν την ίδια σημασία με αυτά της περίπτωσης 1 και τα δύο τελευταία περιγράφουν τη σχέση ξένου κλειδιού με το τρίτο στοιχείο να αντιστοιχεί στην οντότητα που δείχνει το ξένο κλειδί ενώ το τέταρτο στοιχείο στο γνώρισμα της ξένης οντότητας.

Ένα παράδειγμα κωδικοποίησης της οντότητα “Αίτημα” (Request) σε δομή λεξικού φαίνεται στην Εικόνα 3. Παρότι σε ένα πραγματικό υγειονομικό σύστημα η πολυπλοκότητα της βάσης δεδομένων θα ήταν αρκετά μεγαλύτερη και ως προς τις οντότητες αλλά και ως προς τα γνωρίσματά τους η βάση δεδομένων που παρουσιάστηκε παραπάνω περιέχει όλη την απαραίτητη πληροφορία για τη μοντελοποίηση, την υλοποίηση αλλά και την αξιολόγηση της εφαρμογής προγραμματισμού ιατρικών ραντεβού που αυτή η εργασία εξετάζει.



Εικόνα 2: Σχεσιακό σχήμα της βάσης δεδομένων

Η κωδικοποίηση της δομής της βάσης δεδομένων αποτελεί το πρώτο βήμα για την υλοποίηση της διεπαφής μεταξύ της βάσης δεδομένων και του λογικού επιλυτή η οποία θα πραγματοποιεί τη μετατροπή της βάσης δεδομένων σε βάση γνώσης λογικού προγράμματος καθώς σε αυτή τη δομή λεξικού θα βασιστεί η δημιουργία των κατηγορημάτων.

```
'REQUEST': {"ID": ('integer', True),
            "PATIENT_ID": ('text', False, 'PATIENT', 'ID'),
            "TIMESLOT_ID": ('integer', False, 'TIMESLOT', 'ID'),
            "PREFERENCE": ('integer', False),
            "SCORE": ('integer', False),
            "STATUS": ('integer', False)}
```

Εικόνα 3: Κωδικοποίηση της οντότητας "Αίτημα" σε μορφή λεξικού.

Η διεπαφή μεταξύ βάσης δεδομένων και λογικού επιλυτή

Για την επίτευξη της ομαλής ροής δεδομένων μεταξύ της βάσης δεδομένων και του λογικού επιλυτή αναπτύχθηκε μια διεπαφή δύο κατευθύνσεων στη γλώσσα προγραμματισμού Python. Εσωτερικά η διεπαφή αυτή χωρίζεται σε δύο άκρων με το ένα να επιτελεί τη σύζευξη μεταξύ της Python και της βάσης δεδομένων και το δεύτερο να αναλαμβάνει την αλληλεπίδραση της Python με το λογικό επιλυτή, στη συγκεκριμένη περίπτωση, την clingo. Πιο συγκεκριμένα, η υλοποίηση των δύο άκρων της διεπαφής πραγματοποιήθηκε ως εξής:

- Το άκρο επικοινωνίας Python και PostgreSQL υλοποιεί η κλάση **DataModel** κατά την ανάπτυξη της οποίας χρησιμοποιήθηκε η βιβλιοθήκη psycopg2. Ως πρώτη λειτουργία η κλάση επιτελεί τη σύνδεση με μια βάση δεδομένων PostgreSQL (στη συγκεκριμένη υλοποίηση βρίσκεται στο localhost) και την δόμησή της όπως ορίζει ένα δοθέν λεξικό κωδικοποίησης σχήματος όπως αυτό που περιγράψαμε στην προηγούμενη υποενότητα. Αυτή η λειτουργία πραγματοποιείται αρχικά με τη χρήση εντολών δημιουργίας πινάκων (συνάρτηση createTable) και έπειτα με τη φόρτωση δεδομένων από αρχεία τύπου csv (συνάρτηση loadTestData) σε αυτούς. Η κύρια λειτουργία όμως αυτής της κλάσης είναι η υλοποίηση των λειτουργιών άντλησης και αποθήκευσης

δεδομένων από και προς τη βάση δεδομένων (SQL transactions). Αυτές οι συναλλαγές είναι η επιλογή (select), η εισαγωγή (insert), η ενημέρωση (update) και η διαγραφή (delete), οι οποίες υλοποιήθηκαν ώστε να παρέχουν όλες τις εκφραστικές δυνατότητες της SQL, όπως οι συνενώσεις πινάκων (joins), με τρόπο όπου τα ερωτήματα προς τη βάση (queries) να δημιουργούνται και να εκτελούνται με αυτοματοποιημένο τρόπο απαιτώντας μόνο τα ονόματα και τις τιμές των γνωρισμάτων και των πινάκων που εμφανίζονται στο ερώτημα.

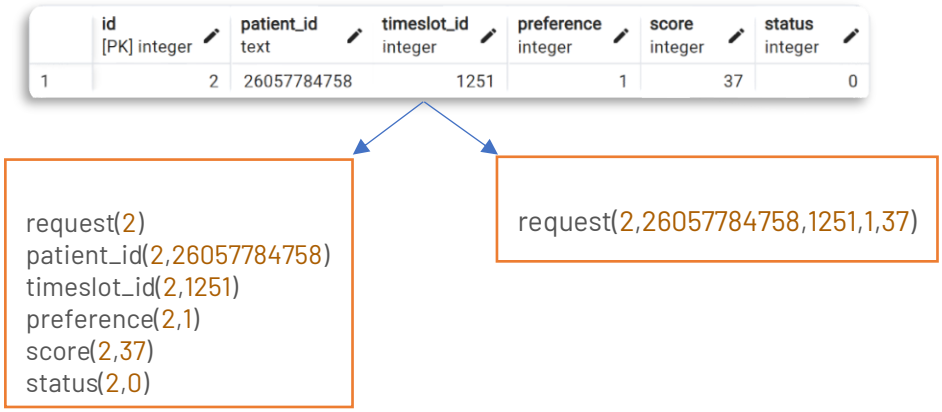
- Το δε άκρο επικοινωνίας μεταξύ της Python και του λογικού επιλυτή υλοποιείται μέσω της κλάσης **KnowledgeBase** η οποία βασίζεται στη βιβλιοθήκη clorm της Python, η οποία υποστηρίζεται απευθείας από την Potassco, επεκτείνοντας λειτουργικά την κλάση FactBase για την αποθήκευση δεδομένων υπό τη μορφή κατηγορημάτων που μπορούν να δοθούν ως είσοδο στο λογικό επιλυτή που παρέχει η βιβλιοθήκη clingo της Python. Σε πρώτο χρόνο η κλάση συνδέεται με μια βάση δεδομένων έχοντας στην κατοχή της τα διαπιστευτήρια σύνδεσης. Έπειτα ακολουθεί μια διαδικασία μετάφρασης των εγγραφών της βάσης δεδομένων από επιλεγμένους πίνακες και γνωρίσματα αυτών σε κατηγορήματα και την εισαγωγή τους σε ένα αντικείμενο τύπου FactBase. Σε επίπεδο αλληλεπίδρασης με το χρήστη η κλάση παρέχει αφαιρετικές διεπαφές οι οποίες παρέχουν τη δυνατότητα χειρισμού των κατηγορημάτων του περιεχομένου της με τρόπο που προσομοιώνει το χειρισμό των εγγραφών σε μια SQL βάση δεδομένων. Ταυτόχρονα περιέχει ενσωματωμένη συνάρτηση η οποία εκτελεί λογικό κώδικα επί του περιεχομένου της βάσης γνώσης παρέχοντας στο χρήστη τον έλεγχο των όλων των βασικών παραμέτρων που εμφανίζονται κατά τη διαδικασία της επίλυσης.

Ειδικότερα, για τη διαδικασία μετάφρασης και κωδικοποίησης των εγγραφών της βάσης δεδομένων στα κατηγορήματα της βάσης γνώσης αξίζει σύντομα να αναλύσουμε τις δύο μεθόδους κωδικοποίησης των κατηγορημάτων εντός της βάσης γνώσης.

1. Η **κατακερματισμένη** κωδικοποίηση (split encoding) η οποία εξετάστηκε πρώτη κατά την εκπόνηση της εργασίας αυτής αποσκοπεί τη μεγιστοποίηση της λεξιλογικής ποικιλίας που μπορεί να χρησιμοποιηθεί κατά την ανάπτυξη λογικών κανόνων και την ελαχιστοποίηση της εμφάνισης ανώνυμων μεταβλητών στα λογικά προγράμματα. Κατά βάση πρόκειται για μία γλωσσολογική προσέγγιση στο πρόβλημα που όχι μόνο μειώνει το χρόνο ανάπτυξης του κώδικα και αυξάνει την αναγνωσιμότητά του αλλά ταυτόχρονα εμφανίζει και τη βέλτιστη υπολογιστική απόδοση λόγω της ακρίβειας στην έκφραση των κανόνων και της απλότητας των κατηγορημάτων που δημιουργεί. Η κωδικοποίηση αυτή υλοποιείται με κατακερματισμό της πληροφορίας έχοντας το κάθε γνώρισμα εντός της βάσης δεδομένων ως λογική μονάδα πληροφορίας. Πιο συγκεκριμένα και όπως φαίνεται στο Σχήμα 2β για κάθε εγγραφή του πίνακα T το πρωτεύον κλειδί κωδικοποιείται ως ένα κατηγόρημα με όνομα T και τιμή την τιμή του πρωτεύοντος κλειδιού της εγγραφής. Έπειτα τα υπόλοιπα γνωρίσματα κωδικοποιούνται ως ένα κατηγόρημα με όνομα το όνομα του γνωρίσματος και έναν συνδυασμό τιμών με την πρώτη να αποτελεί την τιμή του πρωτεύοντος κλειδιού της εγγραφής και το δεύτερο την τιμή του γνωρίσματος.

2. Η **ενοποιημένη** κωδικοποίηση (merged encoding) η οποία υλοποιήθηκε σε δεύτερο χρόνο για τις ανάγκες της εσωτερικής διαχείρισης των κατηγορημάτων από τη βάση γνώσης. Αποδείχθηκε πως ο λογικός κατακερματισμός ανά γνώρισμα παρά τα πολλά του πλεονεκτήματα αυξάνει σε μεγάλο βαθμό την πολυπλοκότητα της εκτέλεσης του συστήματος διαχείρισης της βάσης γνώσης (KBMS, αντίστοιχο του DBMS) το οποίο εκτελεί τις λειτουργίες χειρισμού των κατηγορημάτων προσομοιώνοντας τις συναλλαγές δεδομένων τύπου SQL. Η χρήση αυτής της κωδικοποίησης αυξάνει την απόδοση της εκτέλεσης ερωτημάτων προς τη βάση γνώσης ενώ ταυτόχρονα παρέχει το ελάχιστο πλήθος εγγραφών προς αποθήκευση και δίνει τη δυνατότητα χρήσης ξένων κλειδιών (λειτουργία που δεν προσφέρεται από τη βιβλιοθήκη και υλοποιήθηκε πλήρως σε αυτή την εργασία) και κατ’ επέκταση της πραγματοποίησης σύνθετων

συναλλαγών δεδομένων όπως ερωτημάτων με συνενώσεις (joins) και διασυνδεδεμένων ερωτημάτων (cascade). Το τίμημα εμφανίζεται στην αναγνωσιμότητα και την υπολογιστική ισχύ του κώδικα που αναπτύσσεται με τη χρήση της. Η υλοποίηση αυτής της κωδικοποίησης βασίζεται στην απευθείας μετάφραση μιας εγγραφής σε ένα κατηγόρημα που φέρει το όνομα του πίνακα όπως φαίνεται στο Σχήμα 2γ.



Σχήμα 2: α) Εγγραφή της βάσης δεδομένων. β) Μετάφραση του α) στη βάση γνώσης με χρήση κατακερματισμένης κωδικοποίησης. γ) Μετάφραση του α) στη βάση γνώσης με χρήση της ενοποιημένης κωδικοποίησης.

Στο σημείο αυτό αξίζει να αναφερθεί πως το δίλλημα της επιλογής μεταξύ των δύο αυτών μεθόδων ή ομοιών τους για την κωδικοποίηση των κατηγορημάτων μιας δομής χειρισμού δεδομένων υπό τη μορφή κατηγορημάτων εμφανίζεται συχνά στον τομέα των Γράφων Γνώσης (Knowledge Graphs) με σημαντικότερη εφαρμογή το σύστημα Datalog [9]. Επιπλέον, όλα τα κατηγορήματα υλοποιούνται ως κλάσεις οι οποίες κληρονομούν από την κλάση Predicate της βιβλιοθήκης clorm. Τέλος, για την εκμετάλλευση των πλεονεκτημάτων της κάθε κωδικοποίησης η εφαρμογή χρησιμοποιεί και τις δύο κωδικοποιήσεις. Η ενοποιημένη κωδικοποίηση χρησιμοποιήθηκε για τη λειτουργία του συστήματος διαχείρσης της βάσης γνώσης (KBMS) με τα ενοποιημένα κατηγορήματα να κατακερματίζονται πριν από την εκτέλεση ενός λογικού προγράμματος λειτουργώντας ως είσοδος σε αυτό.

Ανάπτυξη λογικού κώδικα για επαναπρογραμματισμό με ΠΣΑ

Η ανάπτυξη λογικού κώδικα βελτιστοποίησης με το μοντέλο του ΠΣΑ ακολουθεί τα παρακάτω βήματα:

a) Δημιουργία βάσης γνώσης

Το βήμα αυτό υλοποιήθηκε όπως αναλύθηκε στην προηγούμενη υποενότητα με χρήση της κλάσης KnowledgeBase.

b) Παραγωγή σύνθετων κατηγορημάτων

Για την πλήρη περιγραφή του προβλήματος ορίστηκαν τα έξι παρακάτω σύνθετα κατηγορήματα:

1. Το κατηγόρημα **granted** που δηλώνει πως ένα αίτημα έχει γίνει αποδεκτό από το σύστημα.
2. Το κατηγόρημα **appointed** που δηλώνει πως ένας ασθενής κατέχει κάποιο ραντεβού στο σύστημα.

3. Το κατηγόρημα **best** που δηλώνει πως το αίτημα που το ικανοποιεί είναι το πιο ωφέλιμο αίτημα για τον ασθενή που το πραγματοποιεί.
4. Το κατηγόρημα **onlyOption** που σηματοδοτεί πως το αίτημα που το ικανοποιεί είναι το μοναδικό αίτημα του τον ασθενή που το πραγματοποίησε.
5. Το κατηγόρημα **singleRequest** που φανερώνει πως το συγκεκριμένο αίτημα είναι το μοναδικό αίτημα για το ραντεβού που διεκδικεί.
6. Το κατηγόρημα **bestSingleRequest** που δηλώνει πως το αίτημα που το ικανοποιεί είναι το πιο ωφέλιμο από τα αιτήματα που ικανοποιούν το κατηγόρημα singleRequest για τον ασθενή που το πραγματοποίησε.

c) Παραγωγή συνόλων απαντήσεων (answer sets)

Ένα σύνολο απαντήσεων, δηλαδή ένα πρόγραμμα ραντεβού ικανό να ικανοποιήσει τα δεδομένα του προβλήματος, μπορεί να περιέχει τρία είδη αιτημάτων.

1. Αιτήματα που έχουν γίνει **δεκτά** (granted) αλλά **δεν είναι η καλύτερη επιλογή** του ασθενούς που τα έχει πραγματοποιήσει. Η συμπερίληψη αυτών των αιτημάτων στα σύνολα απαντήσεων γίνεται για να δοθεί η δυνατότητα σε έναν ασθενή που ήδη έχει ραντεβού να συνεχίσει να το κατέχει και στο νέο πρόγραμμα.
2. Αιτήματα που **δεν έχουν γίνει δεκτά** (not granted) αλλά είναι **καλύτερες επιλογές** για έναν ασθενή από το ραντεβού που **ήδη κατέχει**. Η συμπερίληψη των αιτημάτων αυτών δίνει τη δυνατότητα σε έναν ασθενή να λάβει ραντεβού καλύτερο από εκείνο που του έχει ήδη δοθεί από το σύστημα.
3. Αιτήματα που ανήκουν σε ασθενή που δεν έχει ακόμα λάβει ραντεβού. Η συμπερίληψη των αιτημάτων αυτών δίνει τη δυνατότητα σε έναν ασθενή που ακόμη δεν εξυπηρετείται από το σύστημα να λάβει κάποιο ραντεβού.

Οι τρεις παραπάνω τύποι αιτημάτων θα αποτελέσουν όρισμα του κατηγορήματος ενέργειας (**grant**) το οποίο θα αναπαριστά και τη δυαδική τους κατάσταση ως εγκεκριμένα ή ως μη εγκεκριμένα. Εάν το κατηγόρημα αυτό εμφανιστεί περιέχοντας ένα ραντεβού τότε το ραντεβού αυτό θα έχει εγκριθεί στο νέο πρόγραμμα. Η διαδικασία παραγωγής των συνόλων που περιέχουν όλους τους δυνατούς συνδυασμούς ενεργειών ονομάζεται αξίωμα ενέργειας (**effect axiom**). Αντίθετα, όλα τα ραντεβού των οποίων η κατάσταση δεν μπορεί να αλλάξει, δηλαδή των εγκεκριμένων αιτημάτων που αποτελούν την καλύτερη επιλογή για τον ασθενή που τα πραγματοποιεί, θα αναπαρασταθούν ως ορίσματα του κατηγορήματος πλαισίου (**claimed**) ενώ η παραγωγή τους θα προκύψει από το αξίωμα πλαισίου (**frame axiom**).

d) Δήλωση περιορισμών ακεραιότητας (integrity constraints)

Στο σημείο αυτό εφαρμόζονται κανόνες ακεραιότητας οι οποίοι αποκλείουν όλα τα σύνολα απαντήσεων που παράχθηκαν στο προηγούμενο βήμα και δεν πληρούν τις παραμέτρους του προβλήματος. Στη συγκεκριμένη υλοποίηση εφαρμόστηκαν τέσσερα είδη περιορισμών ακεραιότητας.

Αρχικά, εφαρμόστηκαν δύο κανόνες διασφάλισης της **λογικής** συνέπειας των συνόλων απαντήσεων με την πραγματικότητα. Πιο συγκεκριμένα ορίστηκε ότι:

1. Κάθε ραντεβού μπορεί να δοθεί μόνο σε έναν ασθενή. Αυτό υλοποιήθηκε ορίζοντας ότι μόνο ένα αίτημα μπορεί να γίνει δεκτό για ένα ραντεβού.
2. Μόνο ένα ραντεβού (από κάθε ειδικότητα στη γενική περίπτωση) μπορεί να δοθεί σε έναν ασθενή.

Ο επόμενος τύπος περιορισμών ακεραιότητας που χρησιμοποιήθηκε είναι οι περιορισμοί για τη διασφάλισης της **δίκαιης λειτουργίας** του συστήματος. Αυτοί συγκεκριμένα ήταν οι εξής:

1. Κάθε ασθενής που έχει ραντεβού στο αρχικό πρόγραμμα θα πρέπει να έχει ραντεβού και το νέο πρόγραμμα.
2. Εάν ένα αίτημα αποτελεί μοναδική επιλογή (onlyOption) για έναν ασθενή τότε αποκλείεται η έγκριση των υπόλοιπων αιτημάτων που παρουσιάζουν χαμηλότερο όφελος από αυτό για το συγκεκριμένο ραντεβού ακόμα κι αν η έγκρισή τους θα οδηγούσε σε μεγαλύτερο κοινό όφελος.

Η τρίτη κατηγορία περιορισμών αφορά τη **μείωση του χώρου καταστάσεων** του προβλήματος αποκλείοντας αιτήματα που δε χρειάζεται να εξεταστούν κατά τη διαδικασία της βελτιστοποίησης. Αυτά τα αιτήματα είναι:

1. Τα αιτήματα που είναι ήδη οριστικά εγκεκριμένα (claimed)
2. Κάθε μοναδικό αίτημα (singleRequest) ενός ασθενή το οποίο δεν αποτελεί το καλύτερό του μοναδικό αίτημα.

Τέλος, η τελευταία κατηγορία περιορισμών ακεραιότητας αφορά στη μεγιστοποίηση του αριθμού των ασθενών που θα εξυπηρετηθούν από το σύστημα παράμετρος που έχει ιδιαίτερο αντίκτυπο στο κοινωνικό σύνολο. Αυτοί οι περιορισμοί είναι οι εξής:

1. Εάν ένας ασθενής έχει πραγματοποιήσει το μοναδικό αίτημα για ένα συγκεκριμένο ραντεβού, το οποίο ταυτόχρονα παρουσιάζει μεγαλύτερο όφελος από το ραντεβού που ο ασθενής ήδη κατέχει τότε το μοναδικό αίτημα αποδίδεται αυτόματα στον ασθενή.
2. Εάν κάποιος ασθενής που δεν εξυπηρετείται από το σύστημα πραγματοποιήσει ένα μοναδικό αίτημα για ένα ραντεβού, τότε το ραντεβού αυτό θα του αποδοθεί αυτόματα.

e) Δήλωση κανόνα βελτιστοποίησης

Τέλος, παρέχεται στο σύστημα ένας κανόνας βελτιστοποίησης ο οποίος καθορίζει τις παραμέτρους προς βελτιστοποίηση και τα κατηγορήματα που τις περιέχουν και των οποίων οι συνδυασμοί θα συγκροτήσουν το βέλτιστο σύνολο απαντήσεων. Στην περίπτωσή μας η παράμετρος είναι το κοινό όφελος και το κατηγόρημα βελτιστοποίησης είναι το κατηγόρημα ενέργειας (grant).

Αξιολόγηση

Η εφαρμογή δοκιμάστηκε σε αρκετά ψευδοτυχαία υγειονομικά συστήματα με ιδιαίτερα υποσχόμενες επιδόσεις. Σε ιδανικές συνθήκες το σύστημα μπορεί να διαχειριστεί προγράμματα έως και 5000 αιτημάτων σε ρεαλιστικό χρόνο όπως φαίνεται στον Πίνακα 4. Λόγω όμως της φύσης του προβλήματος παράμετροι όπως η προηγούμενη κατάσταση του ακυρωμένου ραντεβού, η πολιτική του προηγούμενου προγραμματισμού, η κατανομή αιτημάτων ανά ραντεβού και των αιτημάτων ανά ασθενή μπορούν να επηρεάσουν σε σημαντικό βαθμό το χρόνο εκτέλεσης. Όσο ευνοϊκότερη η τιμή των παραμέτρων αυτών τόσο η διαφορά στο χρόνο εκτέλεσης μεταξύ των δύο κωδικοποιήσεων γίνεται λιγότερο αισθητή. Εκτενέστερη ανάλυση θα γίνει στο κυρίως σώμα της εργασίας.

Ραντεβού	Ασθενείς	Αιτήματα	Χρόνος εκτέλεσης (s) Κατακερματισμένη κωδικοποίηση		Χρόνος εκτέλεσης (s) Ενοποιημένη κωδικοποίηση	
			Best	Worst	Best	Worst
400	500	1000	0.06	2.79	0.09	4.19
960	1200	2400	0.18	152.07	0.15	448.9
2000	2500	5000	421.4	>1800	516.9	> 1800

Πίνακας 4: Χρόνοι εκτέλεσης για τρία ψευδοτυχαία υγειονομικά συστήματα με χρήση κατακερματισμένης και ενοποιημένης κωδικοποίησης

Υλοποίηση υπό πραγματικές συνθήκες

Κάθε εφαρμογή χρειάζεται να υπερβεί περιορισμούς ώστε να μετουσιωθεί από μια απόδειξη μιας ιδέας σε ένα πραγματικό προϊόν. Το πρώτο πρόβλημα που αντιμετωπίστηκε είναι η ανάγκη διαχείρισης πολλών ραντεβού σε δυσοίωνες συνθήκες προγραμματισμού. Στο κύριο μέρος της εργασίας θα παρουσιαστούν αρκετές λύσεις με την ομαδοποίηση των δεδομένων εις βάρος της γενικής βελτιστοποίησης του κοινού οφέλους να αποδεικνύεται ως η πιο αποδοτική και πρακτική. Δεύτερος παράγοντας που η εφαρμογή θα πρέπει σε ένα πραγματικό υγειονομικό σύστημα να διαχειριστεί είναι η ενημέρωση των ασθενών και η λήψη επιβεβαίωσης για το διακανονισμό του νέου τους ραντεβού. Ως λογικότερη λύση για την ομαλή λειτουργία του συστήματος αναδείχθηκε η ασύγχρονη ενημέρωση των ασθενών και ο ορισμός ενός χρονικού ορίου αναμονής για την απάντησή τους. Τέλος, ένα τελευταίο εμπόδιο που η εφαρμογή θα πρέπει να υπερβεί είναι η παρουσία ασθενών διατεθειμένων να εισάγουν ψευδή πληροφορία ώστε να χειραγωγήσουν το σύστημα προς ίδιον όφελος. Αυτό μπορεί να γίνει μέσω της δήλωσης όλων των ραντεβού, κίνδυνος που αποφεύγεται από τον ορισμό ορίου δήλωσης ραντεβού και από την εισαγωγή αποκλειστικά μοναδικών επιλογών ώστε να αποφευχθεί ο οποιοσδήποτε εσωτερικός επαναπρογραμματισμός κίνδυνος που αντιμετωπίζεται με τη χρήση δημοπρασίας δεύτερης τιμής (second-price auction) για τα ραντεβού.

Συμπεράσματα

Φτάνοντας στο τέλος αυτής της σύντομης περιγραφής του περιεχομένου της εργασίας έχει καταστεί φανερή η τεράστια προοπτική που εμφανίζει ο λογικός προγραμματισμός και συγκεκριμένα ο ΠΣΑ στον τομέα της βελτιστοποίησης προβλημάτων με μεγάλο χώρο καταστάσεων και σημαντική πολυπλοκότητα. Αναδεικνύεται παράλληλα η ποικιλία των τρόπων που η Τεχνητή Νοημοσύνη μπορεί να εφαρμοστεί προς όφελος της κοινωνίας. Τέλος, η βασική ιδέα της εφαρμογής μπορεί να χρησιμοποιηθεί για την επίλυση προβλημάτων πέραν του τομέα της υγείας, όπως ο εταιρικός προγραμματισμός συνεδριάσεων, γεγονός που τη καθιστά ιδιαίτερα εύχρηστη και προσαρμόσιμη σε νέα δεδομένα και πεδία εφαρμογής.

Table of Contents

1. Introduction

1.1 About Answer Set Programming

Since its inception in 1972 [1], logic programming has seen significant progress to the point where logic solvers, such as clingo, are capable of handling highly complexity, well-defined, nondeterministic polynomial-time (NP-complete) optimization problems, provided that they are expressed under the semantics of Answer Set Programming (ASP). Several efforts have been made to harness the multiple benefits of ASP, including the high readability of the declarative code due to its resemblance to natural language and the reduced development time due to the optimization algorithm development stage being bypassed. These efforts have been undertaken by fields such as Artificial Intelligence, Bioinformatics, and Databases, making the adoption of this technology a matter of time [10]. Especially in large-scale projects with multi-objective optimization needs, such as digitizing a healthcare system with simultaneous high reliability requirements and usability demands, adopting Artificial Intelligence technologies is a one-way road.

1.2 Modern healthcare

One of the challenges that underscores the limitations of modern healthcare systems in efficiently managing the load assigned to them, particularly in countries where demand far exceeds the capabilities of the medical staff, is the process of managing cancellations in external appointments. Specifically, the management of cancellations with a policy that prioritizes social benefit in optimization raises challenges due to the enormous volume of data in healthcare systems and the critical nature of every decision made since human lives are at stake.

1.3 A technology in need of integration

In this context, this thesis explores the deployment of artificial intelligence, specifically Answer Set Programming (ASP), to implement innovative methods for managing appointment cancellations in modern digital healthcare systems. However, to develop such an application, the creation of an interface between a high-level programming language, in this case, Python, and a logic solver, specifically clingo, was deemed necessary. This interface facilitates the flow of data from the database to the solver and updates the database with the output of the solver. From the early stages of this work, it became apparent that there is a significant gap between ASP and the software industry, often attributed to the lack of interfaces between structural elements that constitute modern software applications, such as databases, and logic solvers. For example, although there have been attempts to use ASP in the field of databases, specifically in optimizing the operation of Database Management Systems (DBMS) [3], there is currently no openly available automated method to convert a database into a knowledge base within a high-level programming environment, which a modern software engineer would realistically choose to implement a project. Simultaneously, there is a noticeable absence of a Knowledge Base Management System (KBMS) equivalent to these database management systems (DMBS).

1.4 Thesis contribution

The final result of this work is a fully functional application for solving the (re)appointment scheduling problem for a healthcare system that utilizes a PostgreSQL database, clingo as the solver, and Python as the communication bridge between the two. This application is intended for use within the data management (backend) section of an online healthcare platform, where citizens will directly interact with the system for their service needs. The Python programming language is used to implement the interfaces between the database and the logic solver, transforming the database into a knowledge base and providing the capability to execute queries and general SQL operations within the knowledge base.

1.5 Structure of the thesis

Section 2 presents the problem of (re)appointment scheduling after cancellations, its actual cost, and the methods used for its management up to this point. Furthermore, for the first time, an AI-based common benefit optimization method is proposed, which constitutes the primary contribution of this work. Section 3 provides a detailed analysis of the application developed for solving the problem. It covers the design of the database, the development of logic code for the solver, and the structure of the two interfaces—one between Python and PostgreSQL and the other between Python and clingo. Finally, it extensively discusses the design of logic code and the concepts it must ensure in a healthcare system. Section 4 evaluates the application based on pseudo-realistic healthcare systems created using Python and the database architecture. The results are promising. The fifth and final section addresses the application's ability to be used in the real world, the challenges it may encounter, and the measures that can be taken to address them.

2. Applications

2.1 Software engineering
2.1.1 Testing, debugging and performance engineering
2.1.2 Information retrieval and data management
2.2 Artificial intelligence
2.3 Bioinformatics
2.4 Robotics and trajectory planning
2.5 Scheduling and timetabling

3. Background

3.1 The ASP paradigm
3.2 Syntax
3.3 Semantics
3.4 Examples
3.4.1 Basics: Node coloring
3.4.2 Planning: Towers of Hanoi
3.4.3 Optimization: Optimal path

3.5 Advantages of ASP

3.5.1 Development

3.5.2 General problem solving

3.5.3 Explainability

4. The medical appointment (re)scheduling problem

4.1 Context

Consider a hospital that serves several thousand patients on a weekly basis and employs dozens of doctors. Let's also assume that each doctor working 8 hours a day provides the system with 8 one-hour appointments from Monday to Friday. In most modern healthcare systems, the process of booking an appointment for a patient works as follows. Initially, the patient either views the current schedule on a platform or contacts the healthcare system's communication center. They are then given an available appointment, which is likely to be scheduled after a certain period of time. In case of cancellation, the appointment becomes unclaimed, and if the cancellation occurs close to the appointment date, it is likely to remain unresolved. This system, although not apparent at first glance, has significant and multi-layered costs, primarily affecting individuals of lower socioeconomic status. Let's consider that the average waiting time for a medical appointment in the USA in 2022 was 26 days [4], forcing 41% of adults to receive delayed treatment for their illnesses, 12% of whom were diagnosed with conditions requiring immediate treatment [5]. At the same time, we cannot ignore the fact that out of the percentage of patients who received delayed treatment, 51.7% had an annual income lower than $50,000, a particularly low income for the USA during that period. Finally, the cost in capital with which unresolved medical appointments burden the U.S. healthcare system amounts to $150 billion [6].

The above data are sufficient to conclude that healthcare systems using the conventional method of appointment management are, to this end at least, dysfunctional and in urgent need of a more efficient approach to reschedule canceled medical appointments. One initial approach that has been proposed is the creation of appointment waiting queues, so that in case of cancellation, the appointment passes to the first eligible patient with the highest scoring request for this appointment. However, as this thesis intends to showcase, this approach is not optimal when the parameter the system seeks to optimize is the common benefit. It is evident that in an approach prioritizing the first-come-first-served basis, individual benefit is maximized, but rescheduling options that could maximize the system's contribution to the overall public without practical harm to the individual are overlooked.

4.2 Solution requirements

Before we proceed with an example, it is necessary to clarify the requirements that a realistic system for rescheduling medical appointments should meet, both conceptually and practically. In terms of design, the system should ensure the following conditions:

1. Every patient who has an appointment in the current schedule should also have an appointment in the new schedule resulting from rescheduling.
2. No patient should receive an appointment with a lower score than what they already have due to rescheduling, as this action would be considered unfair to the individual. Therefore, the system should assign appointments only with higher preference for each patient, ensuring that the individual benefit is not harmed.

As for implementation, the system needs to exhibit the following characteristics:

1. High computational capabilities to handle the massive volume of data in healthcare systems.
2. Technological compatibility with existing software systems used in the healthcare sector.
3. Fairness towards patients, as the stakes involve health and human life.
4. Usability and a pleasant user experience to be accessible to the widest possible segment of society and to motivate users to migrate towards the online appointment management platform.
5. Scalability according to the size of the organization using the system
6. Linguistic capability to describe complex scenarios such as ethical dilemmas that often arise in the healthcare sector.
7. Adequate output capabilities to describe the sequence of rescheduling actions as the system transitions from the initial to the new schedule.

Meeting these requirements would be essential for the development and successful implementation of a system for efficiently rescheduling medical appointments while ensuring fairness and maximizing overall benefit to patients and the healthcare system.

4.3 The proposed solution

To understand the approach proposed by this thesis, it is first necessary to define the quantitative concept of common benefit. For the purposes of this project, common benefit is defined as the sum of the benefit (score) of each granted appointment in the system. The benefit of each appointment is determined by formula (1).

$$Benefit = \frac{0.3}{preference} + 0.7 \cdot priority \quad (1)$$

Preference is defined as the order in which the patient has chosen the specific appointment and inversely affects the benefit value. For example, a patient's first choice will have a preference value of 1, maximizing the benefit. Priority is a parameter derived from the patient's medical history, weighing the factors that make their condition more or less urgent. It is worth noting that this method for calculating benefit is primarily arbitrary, although reasonable for developing an initial implementation. In a future implementation, new parameters can be introduced, and the weights assigned to the two parameters mentioned above can be adjusted.

In a general sense, the proposed approach aims to optimize the allocation of appointments by considering both individual preferences and the urgency of medical conditions, thereby maximizing the overall benefit to the public health system while minimizing inconvenience to individual patients.

To achieve that goal by also satisfying the solution requirements previously mentioned when designing the system, it becomes evident that the problem offers fertile ground for the application of ASP. In the following example, a scenario is described where six patients are requesting an appointment in a schedule containing four appointments. In the initial state, Nikos holds Monday’s appointment, with the waiting queue consisting of George and Kostas in that order. Nikos's decision to

cancel his appointment will trigger rescheduling, which, if a policy that aims to maximize individual benefit is chosen, will result in the appointment being assigned to George since it is of higher preference to him. The program for this policy in the initial state is shown in Table 1, while the program after rescheduling using the conventional individual-centered approach is shown in Table 2 and provides a common benefit equal to 258.

Timeslot	Patient Queue	Request Score
Mon (9 am)	Nikos (Granted)	90
	George	87
	Kostas	86
Tue (1 pm)	George (Granted)	72
	John	60
	Maria	55
Thu (11 am)	John (Granted)	45
	Maria	40
Fri (3 pm)	Kostas (Granted)	71
	Despoina	70

Table 1: Initial schedule

Timeslot	Patient Queue	Request Score
Mon (9 am)	George (Granted)	90
	Kostas	87
		86
Tue (1 pm)	George (Granted)	72
	John	60
	Maria	55
Thu (11 am)	John (Granted)	45
	Maria	40
Fri (3 pm)	Kostas (Granted)	71
	Despoina	70

Table 2: New schedule after conventional rescheduling

Timeslot	Patient Queue	Request Score
Mon (9 am)	Kostas (Granted)	86
Tue (1 pm)	George (Granted)	72
		60
		55
Thu (11 am)	John (Granted)	45
	Maria	40
Fri (3 pm)	Despoina (Granted)	70

Table 3: New schedule after AI rescheduling

This example illustrates how ASP can be applied to reschedule appointments, considering both individual preferences and priority while maximizing the common benefit provided by the healthcare system.

Based on the principles mentioned above, the main proposal of the new method suggested by the common-benefit approach to the problem is encapsulated in the following statement:

The non-selection of the first available person for the assignment of a canceled appointment is acceptable if the patient already has an appointment, and this choice, through a chain of actions, leads to a new schedule with a greater common benefit.

Applying the above axiom to the schedule in Table 1 and foreseeing the maximum common benefit that this choice will bring, we choose not to assign Monday's appointment to George after Nikos's cancellation. Instead, the appointment will be given to Kostas, and George will retain the Tuesday appointment. Following this logic, no patient is unfairly treated immediately, and the new schedule after rescheduling, as shown in Table 3, exhibits a common benefit equal to 273, making it the optimal solution for the specific initial situation.

4.4 Complexity expression

However, if we realize that the complexity of the problem is exponential with respect to the number of requests from patients for the total number of appointments, as shown in Equation 2, we are inevitably faced with the following questions:

1. How can we identify the optimal schedule in terms of common benefit in such a large state space?
2. What technology could consider all the parameters to ensure the requirements set for rescheduling while simultaneously executing the optimization process in a reasonable amount of time, making it a realistic solution for a large-scale healthcare system?

The two questions above are what led to the choice of AI for solving the problem because, in addition to the many additional benefits it offers in such an application, the approach of artificial intelligence, and specifically knowledge representation, provides a holistic approach to the various parameters and a computationally feasible solution to handle its high complexity. Specifically, the complexity of the problem is given as follows:

$$O(S^R) = O\left(S^{d \cdot T}\right) = O\left(S^{d \cdot D \cdot h \cdot t}\right) = O(2^{d \cdot D \cdot h \cdot t}) \quad (2)$$

Where S represents the possible states that a request for an appointment can take, and R is the number of requests in the system. Since each request can either be accepted or remain in the waiting queue, the state of a request is essentially a binary logic variable with values true (granted) or false (not granted). As a result, the basis of the exponential complexity will be 2. The parameters d and T are used to relate the number of requests to the actual variable of the problem, which is the number of available appointments in the system. Demand, represented by d, signifies the number of requests a patient is allowed to make, while the doctor's working hours T refer to the number of appointments that the system can provide. These are obtained by multiplying the number of doctors D by their daily working hours h and the time span covered by the schedule in days, denoted by t.

5. Methodology

5.1 An overview

Having conducted a thorough analysis of the problem, we proceed to describe the software application that implements the proposed approach for the rescheduling of

medical appointments. This system consists of two primary data flow nodes and an interface that serves as a communication channel between them. Data, as in any modern software application, are stored in a database and retrieved from it whenever the rest of the system requires them. PostgreSQL was the technology used to implement the database, considering that this application is intended for use within the backend of a web application. The Python programming language, specifically the psycopg2 library, was used to connect the database to the corresponding data node of the database. In the second data flow node of the system resides the logic solver, which in this specific case is the Clingo solver. This node is where all data processing occurs in order for the new optimal appointment schedule to be acquired after a cancellation. The interface between these two nodes facilitates the transfer of data from the database to the logic solver and the return of results back to the database by updating the corresponding records. In summary, this system enables efficient data processing and the generation of optimal appointment schedules.

5.2 The database

5.2.1 Design

A well-designed database is essential for the development and evaluation of any modern application. In this project, the relational model was chosen for the database design, and SQL technology was used for its implementation. This approach makes the structure of the database more evident and understandable to the reader, while also facilitating the translation of the database into a knowledge base due to the direct logic transition from records to predicates. The database, whose entity-relationship diagram is presented in Figure 1, simulates the fundamental aspects of a real healthcare system, allowing testing of the application's computational capabilities under realistic conditions. The final relational schema for a complete description of the database structure can be found in Figure 2. The ERD Plus application [7] was used for developing the entity-relationship diagram, while the DB designer [8] application was used for creating the relational schema.

5.2.2 High-level structural representation

From the initial stages of the work's development, it became evident that a method of encoding the database using a data structure provided by high-level programming languages like Python was necessary for generating pseudo-realistic data for system testing and constructing the knowledge base. The selected structure is that of a multi-level dictionary (multilevel hash map), with the keys of the first level representing the entities of the relational schema. Each key's value in the first level is a new dictionary with keys representing the attributes of each entity, and the values being lists containing the necessary information for representing the attribute properties. The encoding of the database structure is the first step in implementing the interface between the database and the logic solver, which will perform the conversion of the database into a logic program knowledge base, as this structure will be the foundation for the creation of the knowledge base predicates. A first-level key-value pair of this multilevel hash map for the ‘Request’ entity of the database can be seen below in Figure 3.

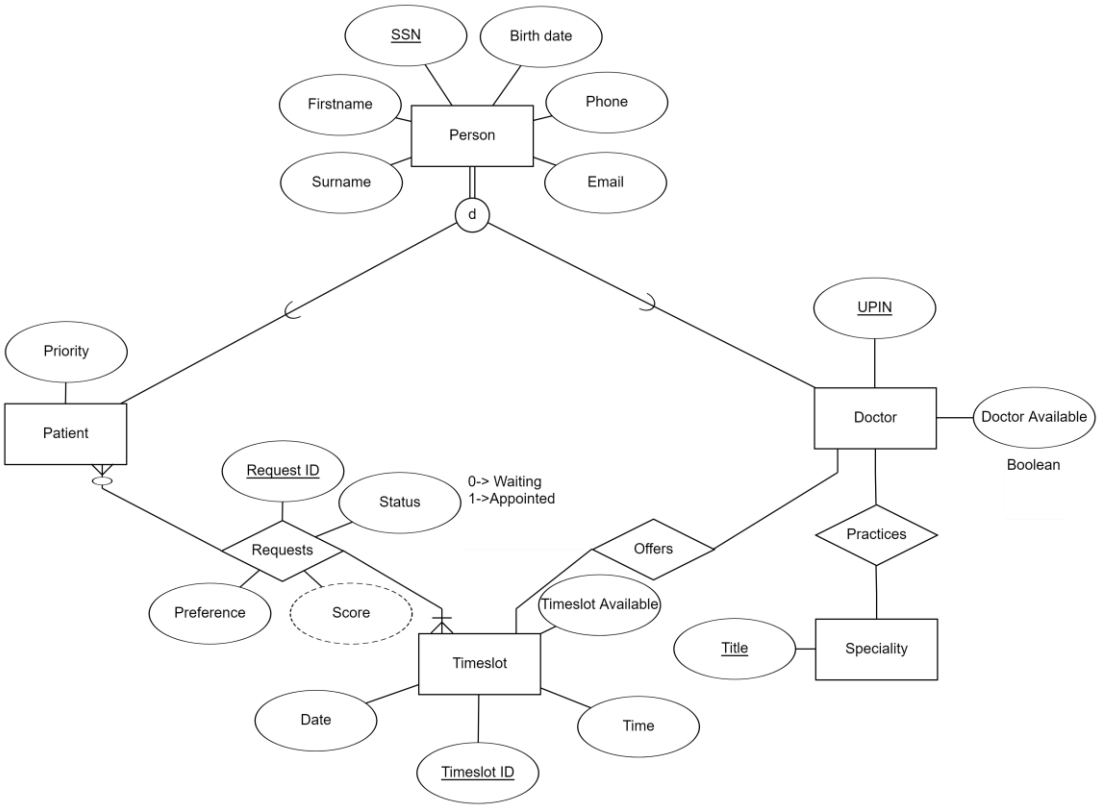


Figure 1: Database ERD

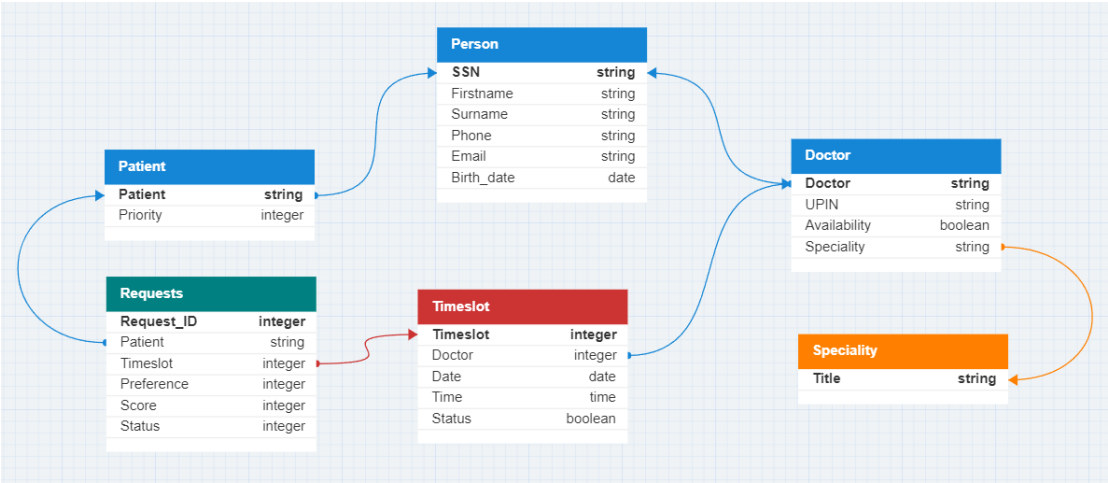


Figure 2: Database schema

```
'REQUEST': {"ID": ('integer', True),
            "PATIENT_ID": ('text', False, 'PATIENT', 'ID'),
            "TIMESLOT_ID": ('integer', False, 'TIMESLOT', 'ID'),
            "PREFERENCE": ('integer', False),
            "SCORE": ('integer', False),
            "STATUS": ('integer', False)}'REQUEST': {"ID": ('integer', True),
            "PATIENT_ID": ('text', False, 'PATIENT', 'ID'),
            "TIMESLOT_ID": ('integer', False, 'TIMESLOT', 'ID'),
            "PREFERENCE": ('integer', False),
            "SCORE": ('integer', False),
            "STATUS": ('integer', False)}
```

Figure 3: The "Request" entity in dictionary form

5.3 The solver - database interface

For the smooth data flow between the database and the logic solver, a two-way interface was developed in the Python programming language. Internally, this interface is divided into two ends, with one end handling the connection between Python and the database, and the other end responsible for the interaction between Python and the logic solver, in this case, clingo. Specifically, the implementation of the two ends of the interface is as follows:

1. The Python-to-PostgreSQL communication end is implemented by the DataModel class, which uses the psycopg2 library. This class first establishes a connection to a PostgreSQL database (in this implementation, it is set on localhost) and sets up the database structure based on a given schema encoding, as described earlier. Its main function is to implement data retrieval and storage operations between Python and the database using SQL transactions, including select, insert, update, and delete operations. These transactions are designed to offer all the expressive power of SQL, allowing queries to be generated and executed automatically, requiring only the names and values of attributes and tables in the query.

2. The interface end between Python and the logic solver is implemented through the KnowledgeBase class, which relies on the clorm library for Python. This class extends the functionality of the FactBase class for storing data in the form of predicates that can be input to the logic solver provided by the clingo library in Python. Initially, this class connects to a database, using the provided credentials. Then, it proceeds to translate records from the database into predicates and inserts them into a FactBase object. For the user interaction, the class provides abstract interfaces that allow predicate manipulation similar to how records are handled in an SQL database. It also includes an integrated function that executes logic code on the contents of the knowledge base, providing users with control over key parameters during the resolution process.

5.3.1 The Data model

The **DataModel** class was developed to enable the interaction between the interface and the database in an abstract level. The class constructor receives the database credentials and generates a Connection object provided by the **psycopg** Python module. Using this connection, the class offers a variety of methods to interact with the database from the creation and deletion of the database tables to the execution of dynamic SQL queries. The database tables are created based on the database schema, a multilevel dictionary provided to the constructor, using the **createTables** function and are populated with records using the **loadData** method from preexisting csv files.

After the creation of the database the user can execute dynamically created SQL queries which are sanitized through symbolic replacement inside the **execute** method of a cursor object. The query results are returned as a list of lists that include the data of each record. What is noticeable about this class is the dynamic generation of the queries meaning there is no need for the user to develop the queries to be executed every time a database transaction has to occur. Instead the user only has to call the appropriate function of the interface and enter the query parameters, namely the tables affected by the query, the attributes that constitute the query conditions and their values. The structure of the DataModel class can be seen in Figure 4.

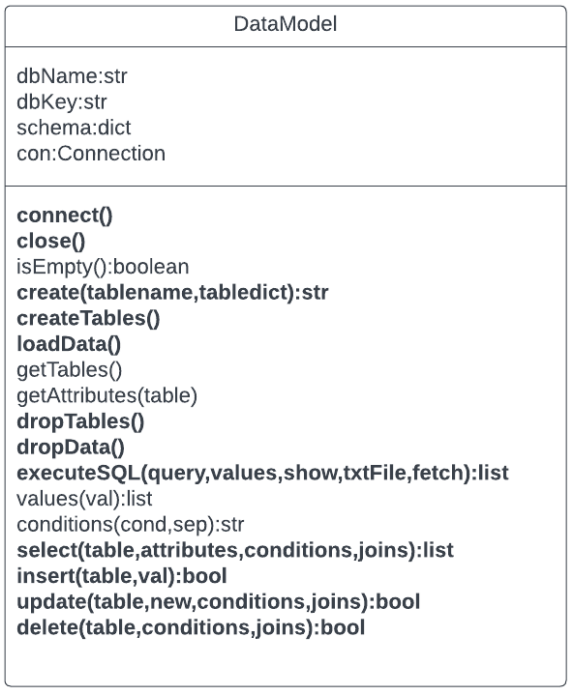


Figure 4: UML diagram for the DataModel class

5.3.2 From record to predicate

Regarding the encoding and encoding methods of the data from the database into the knowledge base, two methods were used:

1. **Split Encoding**: This method aims to maximize lexical variety that can be used during the development of logic rules while minimizing the appearance of

anonymous variables in logic programs. It is a linguistic approach that not only reduces code development time and enhances readability but also improves computational performance due to the accuracy in expressing rules and the simplicity of the generated predicates. In this method, each attribute within the database is treated as a logic information unit. For each record in table T, the primary key is encoded as a predicate with the name T and the value of the primary key as its value. Then, the remaining attributes are encoded as predicates with names corresponding to their attribute names and values as arguments.

2. **Merged Encoding**: This method was implemented later for internal management of predicates from the knowledge base. It was found that the logic fragmentation per attribute, despite its advantages, significantly increased the complexity of the knowledge base management system (KBMS), which simulates data handling operations like SQL transactions. This method increases the performance of querying the knowledge base while also reducing the number of records to be stored. It allows the use of foreign keys (a feature not provided by the library and fully implemented in this work) and, by extension, the execution of complex data transactions, such as queries with joins and cascaded queries. However, it may affect code readability compared to split encoding. This method directly translates a record into a predicate named after the table, as shown in the provided schema.

These methods provide different trade-offs between computational performance and code readability, and the choice between them depends on specific project requirements and goals.

At this point, it's worth mentioning that the dilemma of choosing between these two encoding methods, or similar ones, for encoding the predicates of a data handling structure in the form of predicates, is common in the field of Knowledge Graphs, with the most notable application being the Datalog system. Additionally, all predicates are implemented as classes that inherit from the Predicate class in the clorm library. Finally, to leverage the advantages of each encoding method, the application uses both encodings. The merged encoding is used for the operation of the Knowledge Base Management System (KBMS), with the merged predicates being fragmented before executing a logic program, serving as input to it. This hybrid approach allows the application to benefit from the strengths of both encoding methods while managing the knowledge base efficiently. It combines the ease of handling and the expressiveness of the split encoding with the performance improvements and capabilities of the merged encoding when interacting with the logic solver. This flexibility ensures that the system can meet a wide range of requirements and effectively manage data in various scenarios. In summary, the choice of encoding method and the hybrid approach reflect the careful consideration of trade-offs between readability and computational performance, allowing the system to perform well while maintaining code clarity and adaptability.

5.4.3 Knowledge base implementation

One of the first functions that had to be implemented during the development of this project was the creation of a fully functional, SQL like Knowledge Base directly from the contents of a database with an automated process. To this end, a KnowledgeBase Python class was developed providing many functionalities to its user. At first, the KnowledgeBase can be populated either by a database if its credentials are provided during the instance creation or by a list of data dictionaries. When linked to a database, the KnowledgeBase instance can either constitute a

direct copy of the database or contain a subset of its data. Both entities and attributes to be requested from the database can be specified a nd conditions to the requested data can also be applied. As an example, only the available timeslots are requested for each rescheduling from the database. The goal is to only transfer the data necessary for rescheduling each time from the database to minimize the database size and simultaneously decrease the amount of information to be transferred from one node of the interface to the other.

After the creation of the KnowledgeBase instance the knowledge base predicates are created using a merged encoding based on the database schema dictionary. The predicates are created as Python classed using Python's dynamic typing capabilities. These classes are stored in a dictionary to be instantiated whenever a new record is inserted into the knowledge base and use inheritance to claim all the properties of the Predicate class from the clorm library. Afterwards, the KnowledgeBase connects to the database with the **bind2db** using the credentials given to the constructor and the data flow from the database to the knowledge base node starts through the **db2kb** function. This function uses the select query of the DataModel class to request the data as a list of lists where each element contains the data of a record in the exact order stated in the query. Furthermore, the class implements all the SQL like transactions providing mechanisms to support advanced SQL features such as foreign keys, joins and cascade option in deletion and update queries. Simultaneously, the class includes a variety of methods for internal management of the knowledge base content such as the transformation of merged to split predicates, the extraction of a part of the knowledge base, the reload of data from the database and the return of the knowledge base to its initial state. Finally, the class offers a run method for executing logic code with its content as the knowledge base of the logic program which will be thoroughly describes in the next subsection.

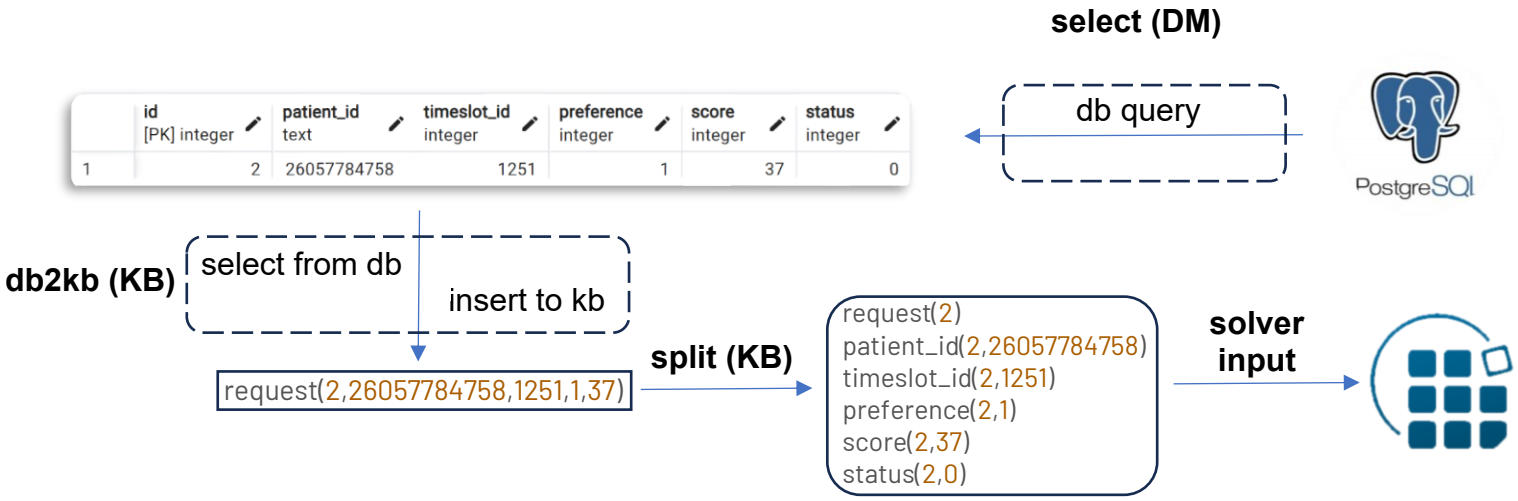


Figure 5: Transformation of a database record passing through the interface

5.4.4 Executing ASP logic programs in Python

In Python there are two main ways to execute logic programs. The first and less practical is the execution of clingo code with the program in string format. This means that the knowledge base has to be also in string format making it almost

impossible to manage efficiently in a scenario with large volumes of data as input to the logic program. This way of executing logic code is provided by the clingo module. The other and more elegant approach on executing logic code in Python is provided by the clorm module. Resembling the philosophy of a web framework the solver provided by the clorm module uses predeclared types, namely classes that extend the Predicate class of the module. Afterwards, these predicates are inserted into a FactBase instance, which is linked to a clorm Control object. Simultaneously, the path the ASP code is provided to this Control object followed by the grounding step. Before solving the problem an on_model class has to be implemented containing a nonlocal solution variable which will receive a new with every answer set that is found to satisfy the problem during the solving process. At the end of this process, which is executed by calling the solve function of the Control object, the solver returns the final (and in this case the optimal) answer set along with a dictionary containing thorough statistics describing the solving process such as the cost and the CPU execution time. This highly modular and structured approach is considered necessary to develop large scale applications using ASP especially in an era when frameworks have overtaken the software industry.

Interface description

Input: Database name, schema, credentials, and conditions
Solver parameters: Path to solver, output predicates, time and model constraints, input data subset and the type of encoding
Output: The optimized answer set and optimization statistics

Data Fabricator pseudo code for entity e

1: Run for 'quantity' times
2: Run for every entity attribute
3: Declare a dictionary representing the final record with the attributes as keys and their values as values, call it temp_dict
4: If the attribute is foreign load the corresponding foreign key from the foreign entity
5: Choose a foreign key, if the attribute is a primary key also remove it from the dataset acquired by the previous step
6: Add the key value pair to temp_dict
7: If the attribute is not foreign
8: Create a random value with faker or for specific attributes choose a value from a predetermined set
9: Add the key value pair to temp_dict
10: Append the temp_dict to a list of dictionaries containing all the records
11: Handle the status of the requests (Only for the Request entity)
12: Write the list of dictionaries to a .csv file

Data Model use case

```
1: db = DataModel(postgreskey, dbname, dbkey, schema) # Connect to the data base
2: db.dropTables() # Clear the previous table structure to recreate with new schema (optional)
3: db.dropData() # Clear all the previous data from the database (optional and only useful if the schema remains the same)
4: fab = DataFabricator(schema, minimum_people, maximum_people, minimum_doctors, maximum_doctors,
minimum_specialites, maximum_specialites, demand, timeslot_availability, tSpan=10, seed=1) # Generate new data
5: db.loadData() # Load the data created by the Fabricator instance form the .csv files to populate the database
6: data = db.select(entity, attributes, conditions, joins) # Execute a query such as a select command
```

Knowledge Base use case

```
1: kb = KnowledgeBase (name, schema, credentials, conditions) # Create the knowledge base
2: kb.toFile() # Export the knowledge base as a text file for a better data overview (optional)
3: # Create output auxiliary predicates to express the solution
4: solution = kb.run (path, outPreds, searchDuration, models, subKB, merged) # Run an ASP program on the KB data
5: # Process the solution data (optional)
6: kb.update(entity, conditions, values) # Update KB and DB with the results of the optimization process
```

Figure 5: A brief documentation of the developed interface including the data generation and the database creation.

Figure 6: UML class diagram for the KnowledgeBase class

5.5 Developing the ASP logic code

The development of logic optimization code using the Answer Set Programming (ASP) model follows the following steps:

i. **Knowledge Base Creation**:
This step involves creating a knowledge base using the KnowledgeBase class, as discussed in the previous subsection.

ii. **Generation of Complex Predicates**:
To fully describe the problem, six complex predicates are defined:

1) The "**granted**" predicate indicates that a request has been accepted by the system.
2) The "**appointed**" predicate signifies that a patient has an appointment in the system.
3) The "**best**" predicate indicates that the request satisfying it is the most beneficial for the patient making it.
4) The "**onlyOption**" predicate signifies that the request satisfying it is the only request for the patient who made it.
5) The "**singleRequest**" predicate reveals that a specific request is the only request for a particular appointment.
6) The "**bestSingleRequest**" predicate states that the request satisfying it is the most beneficial among requests satisfying the "singleRequest" predicate for the patient who made it.

iii. Answer Set Generation
An answer set, which represents a program capable of satisfying the problem's data, can include three types of requests:

1) Requests that have been **granted** but are **not** the **best** option for the patient who made them. These requests are included to allow a patient who already has an appointment to continue holding it in the new program.
2) Requests that have **not** been **granted** but are **better options** for a patient than the appointment they already have. Including these requests allows a patient to receive a better appointment than the one originally assigned by the system.
3) Requests belonging to patients who have not yet received an appointment and can thus be characterized as **not appointed**. Including these requests allows patients who are not yet served by the system to receive an appointment.

The process of generating sets that contain all possible combinations of actions is called an "effect axiom." Conversely, appointments whose status cannot change, such as approved requests that are the best option for the patient, are represented as arguments of the "claimed" predicate, and their generation is derived from the "frame axiom."

iv. Declaration of Integrity Constraints
At this stage, integrity rules are applied to exclude all answer sets generated in the previous step that do not meet the problem's parameters. Four types of integrity constraints are applied:

Initially, two rules were applied to ensure the logic consistency of the answer sets with reality.

1) Each appointment can be assigned to only one patient, ensuring logic consistency.
2) Each patient (in general) can have only one appointment, maintaining logic consistency.

To ensure the fair operation of the system, two additional constraints are imposed:

1) Every patient who has an appointment in the initial program must have an appointment in the new program.
2) If a request represents the only option for a patient, it must be automatically approved.

To reduce the state space of the problem, requests that do not need to be examined during optimization are excluded:

1) Requests that are already definitively approved (claimed).
2) Unique requests (singleRequest) for a patient that are not their best request.

v. **Declaration of Optimization Rule:**
Finally, an optimization rule is provided to determine the parameters to optimize and the predicates that contain them. In this case, the parameter is the common benefit, and the optimization predicate is the "grant" predicate. This logic optimization code helps in finding the optimal set of appointments considering various constraints and parameters.

6. Evaluation

6.1 Data fabrication

6.2 Testing

6.3 Performance variables

7. Real-world implementation

7.1 Scalability

7.1.1 Optimization methods

7.1.2 Implementing batching

7.2 Managing action chains

7.3 Robustness

8. Conclusion

Bibliography

[1] V. Lifschitz, Answer Set Programming, Texas: Springer, 2019.

[2] H. Hoos, R. Kaminski, M. Lindauer and T. Schaub, "aspeed: Solver Scheduling via Answer Set," 2013.

[3] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri and F. Scarcello, "The DLV System for Knowledge Representation," 2003.

[4] M. Hawkins, "Survey of Physician Appointment Wait Times and Medicare and Medicaid Acceptance Rates," AMN Healthcare, 2022.

[5] A. H. Gertz, C. C.Pollack, M. D.Schultheiss and J. S. Brownstein, "Delayed medical care and underlying health in the United States during the COVID-19 pandemic: A cross-sectional study," *Preventive Medicine Reports,* 2022.

[6] J. Gier, "Healthcare Innovation," SCI Solutions, [Online]. Available: https://www.hcinnovationgroup.com/clinical-it/article/13008175/missed-appointments-cost-the-us-healthcare-system-150b-each-year. [Accessed August 2023].

[7] N. Jukić, S. Vrbsky, S. Nestorov and A. Sharma, "ERD Plus," 2015. [Online]. Available: https://erdplus.com/. [Accessed 2023].

[8] J. Perez, "DB Designer," 2006. [Online]. Available: https://erd.dbdesigner.net/. [Accessed 2023].

[9] M. Krötzsch, "Knowledge Graphs - Lecture 8: Limits of SPARQL," slide 17, 15th Dec 2020.

[10] G. Brewka, T. Eiter and M. Truszczyński, "Answer Set Programming at a Glance," 2011.

[11] O. E. Khatib, "Job shop Scheduling under Answer Set Programming Environment," *International Journal of Engineering and Innovative Technology (IJEIT),* vol. 5, no. 5, 2015.

[12] C. Dodaro and M. Maratea, "Nurse Scheduling via Answer Set Programming," 2017.

[13] C. Dorado, G. Galatà, M. Kamran, M. Maratea and I. Porro, "Operating Room (Re)Scheduling with Bed," 2021.

[14] M. Alviano, C. Dodaro and M. Maratea, "Nurse (Re)scheduling Via Answer Set Programming," 2014.

An AI Approach to Large-Scale Medical Appointment (Re)Scheduling Using ASP

Stavros Kanias

University of Patras, University Campus Rion, Patra, GR26504, Greece

Abstract

Despite their significant solving capabilities ASP solvers have not yet crossed the line between academic and widespread industrial use. Even in areas where classical AI solutions developed in the ASP framework exceed solvers that utilize procedural algorithms in terms of raw performance, the latter approach is usually preferred. The lack of integration between ASP solvers and standard software development components, such as SQL databases, is commonly cited as a reason for this tendency. To this end, a Python interface for converting an SQL database (where the healthcare system's data reside) to a logic program knowledge base was developed[2]. This interface supports all the basic SQL transactions adding a level of abstraction on top of the Potassco backed CLORM[3] module. Simultaneously, a new approach to medical appointment (re)scheduling is proposed taking into consideration the total community benefit, showcasing the immense potential of ASP in the modern software industry. An application that uses the ASP-SQL interface to solve the medical appointment (re)scheduling problem with this approach is the final product of this research.

Keywords

Answer Set Programming, Appointment Rescheduling, Common Benefit Maximization, Clingo, Python

Introduction

Since its inception in 1972 [1], logic programming has developed to the point where contemporary solvers, such as clingo, are able to solve complex, mostly NP-complete [11], optimization problems when they are expressed under answer set semantics. Based on those capabilities and the fact that it can greatly reduce development time due to its declarative nature, ASP has been successfully used in several research areas, including Artificial Intelligence, Bio-informatics , and Database querying while recently entering the software industry as a viable software alternative to procedural algorithms [12]. Specifically, modern healthcare systems pose a plethora of challenges in the road to full digitalization and the improvement of patient experience, one of which is the management of appointments in a way that maximizes the community benefit. This challenge arises from the sheer scale of data that characterizes a healthcare system and the ethical implications that come to the forefront when a patient's health is at stake. In fact, digital systems are essential to overcome this challenge and as the paper proposes, AI and in particular ASP, can provide innovative perspectives and new capabilities to modern digital healthcare systems. Before solving the

ENIGMA-23, September 03-04, 2023, Rhodes, Greece

up1066563@upnet.gr

0009-0003-2328-9660

CEUR Workshop Proceedings (CEUR WS.org)

[2] The source code for the application can be found on https://github.com/StavrosKanias/Medical_Appointment_Rescheduling_App

[3] At the time of writing CLORM is an open-source project with its source code residing on https://clorm.readthedocs.io

medical rescheduling problem with AI reasoning a technical challenge must be overcome. The gap between ASP solvers and the software industry, since every modern application consists of many separate components interacting with one another, is a phenomenon commonly attributed to the absence of abstract interfaces between ASP solvers and modern high-level languages. For example, although few attempts to integrate SQL databases with logic programs have been made [3] an automated solution for converting an SQL database into an ASP knowledge base in a high-level programming language environment has yet to become available. Thus, in this paper the structure and use of an interface between a PostgreSQL database and a clingo logic program developed to simplify the data flow between the database and the solver is presented.

The final software brings the possibility of using ASP solvers in the backend of industrial scale software applications to the spotlight. The Python programming language is used for the interface development utilizing the Predicate class of the Potassco backed CLORM module to encode the database records and the Control class to execute the logic program on the derived knowledge base. In Section 2 a more in-depth examination of the medical appointment (re)scheduling problem is presented followed by a simple example comparing a classical to an AI approach for attaining the optimal solution to the problem. Section 3 contains a structural and logical analysis of the ASP rescheduler program. First, the structure of the SQL database is presented. The second subsection is dedicated to the structure and usage of the Python interface. In the third subsection, the code structure rule content of the logic program used to solve the problem is thoroughly analyzed. Furthermore, in Section 4, various statistics describing the performance of the solution for different datasets are presented and comments are made on the factors that affect the program's execution time. Finally, in Section 5, possible implications of a real-world application based on this solution are presented.

The Appointment (Re)Scheduling Problem

Consider a hospital with thousands of patients and dozens of doctors. Each doctor works for 8 hours daily providing 8 one-hour appointment timeslots per day from Monday to Friday. In most contemporary healthcare systems patients can book an appointment only if it is currently available. As soon as a vacant timeslot is requested it is granted by the system to the patient preventing all the other patients from requesting the same timeslot. The problem that arises from this policy is that in the case of a cancellation the timeslot will again become available but will most likely be wasted since no waiting queue was formed for another patient to claim it instantly. Even in the case where a waiting queue exists a first-come-first-served approach will be followed, and the system will grant the request with the highest score for the timeslot in question. In this case, given that the patients would freely choose a better timeslot for themselves if asked, the community optimal assignment may be missed. As described in the following example, using an AI approach to maximization, if the system can choose not to give the canceled appointment to the highest-scoring request of the queue, a chain-reaction of rescheduled appointments can occur through which a much higher community benefit can emerge. A scenario that allows patients to select more than one timeslot with ascending preference order, where 6 patients claim 4 timeslots can be seen in Table 1.

Table 1
Initial Schedule

Timeslot	Patient Queue	Request Score
Mon (9 am)	Nikos (Granted)	90
	George	87
	Kostas	86
Tue (1 pm)	George (Granted)	72
	John	60
	Maria	55
Thu (11 am)	John (Granted)	45
	Maria	40
Fri (3 pm)	Kostas (Granted)	71
	Despoina	70

The score of each request is calculated as the weighted sum of two factors, the request's preference, and the patient's priority. As preference we define the request's temporal order among all the patient's requests. For example, the first request made by the patient to the system will be registered with priority 1, the next with priority 2 etc. The second factor, priority, is considered as a combined metric of the patient's health calculated by the system according to the patient's history. It is used to determine the urgency attributed to a particular request. In the current implementation the scores are derived from the following formula.

$$Score = \frac{0.3}{preference} + 0.7 \cdot priority \quad (1)$$

In the scenario presented in Table 1, after defining the community benefit as the sum of the scores of the granted requests, a community benefit of 278 is offered by the healthcare system to society. In the case where Nikos cancels his appointment, the first timeslot becomes unclaimed. If a first-come-first-served approach is used the timeslot will be granted to George and thus a community benefit of 258 will emerge. In contrast the AI approach previously mentioned and described in depth in the next section will create the schedule seen in Table 2 which yields a higher community benefit of 273.

```
'REQUEST': {
"ID": ('integer', True),
"PATIENT_ID": ('text', False, 'PATIENT', 'ID'),
"TIMESLOT_ID": ('integer', False, 'TIMESLOT', 'ID')
"PREFERENCE": ('integer', False),
"SCORE": ('integer', False),
"STATUS": ('integer', False)
}
```

Table 2
Schedule after AI rescheduling

Timeslot	Patient Queue	Request Score
Mon (9 am)	Kostas (Granted)	86
Tue (1 pm)	George (Granted)	72
		60
		55
Thu (11 am)	John (Granted)	45
	Maria	40
Fri (3 pm)	Despoina (Granted)	70

Developing an AI solution using SQL, Python and ASP

Database design

A thoughtfully designed and implemented database was necessary to develop and test the clingo – database interface. The relational model was chosen for the design to make the structure of the database more obvious. Since the interface is developed with backend applications in mind the PostgreSQL environment was used alongside the Python psycopg2 module. Simulating a typical hospital application, the database was structured as can be seen in Figure 2. This diagram was transformed into the final database schema seen in Figure 3 and translated to Python code as a dictionary of dictionaries.

To be more specific, the database schema in the source code is a dictionary with the keys being the entities and the values being subdictionaries with the entity's attributes as keys and tuples of two to four elements as their values. Each value follows the following convention, the first element is the type of the attribute, and the second element is a boolean signifying if the attribute is a primary key. The third element (if it exists) signifies the uniqueness of the attribute in a three-element tuple or the foreign entity that this attribute points to if it is a foreign key. Finally, the fourth element appearing only in foreign keys is the foreign attribute to which the attribute refers to. An example of this conversion methos for the request entity can be seen in Figure 1.

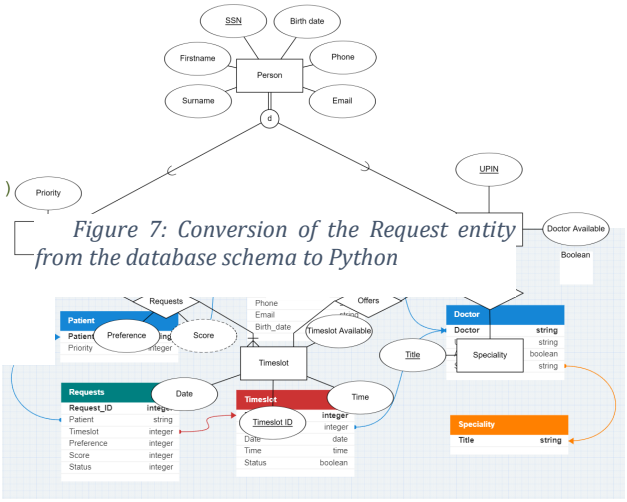


Figure 7: Conversion of the Request entity from the database schema to Python

Although in a real healthcare system the complexity of such a database would be much higher, only the request, timeslot and doctor entities are needed to feed the clingo rescheduler with data to output a general solution for the healthcare system as can be seen in the following section, making every other information stored in the database irrelevant.

Figure 3: Database relational schema [8]

Figure 2: Database Entity Relationship Diagram [7]

Python interface

To fuel clingo with large scale data efficiently an interface was developed in the Python programming language. The functionalities required to enable the data flow between a database and the clingo solver were implemented as a KnowledgeBase Python class. This class instantiates a knowledge base structure that extends the FactBase class from the Potassco backed CLORM framework. The goal of the interface is to create a one-to-one copy of the database in a format intended to function as input for the solver provided by the Control class of the clingo Python module. A typical use case of the interface can be seen in Figure 5.

For the conversion of the database data to clingo predicates, two approaches were taken into consideration, each with its own strengths and weaknesses. Both extensively use Python's dynamic typing capabilities for creating the KB predicates based on the database schema of Figure 3. An example of this record to predicate

translation for both encodings can be found in Figure 4.

These predicates extend the Predicate class found in the CLORM framework specifying each time the name and type of each database attribute. At this point it is useful to mention that clingo only supports integers as numerical values and thus the fabricated data used to test the application had to be adjusted to this constraint. Another challenge that arose while attempting to preserve the database properties in the knowledge base was the absence of the concept of foreign keys which was manually implemented.

To this end the merged approach was implemented intending on keeping the information of each database record unified in one predicate and thus enabling the use of joins and other SQL capabilities based on matching foreign keys. Simultaneously it achieves minimization of the number of predicates in the knowledge leading to faster query execution times. Although the ability to solve the problem utilizing the merged encoding is given to the user (the ASP program must conform with the knowledge base encoding) the execution time for the optimization process is lacking behind compared to the second encoding. Another problem with this approach in the context of ASP code development is the creation of logic programs with many anonymous variables making them less readable and intuitive and simultaneously complicating the expression of more advanced logical rules.

```
request(2,
26057784758,
```

	id [PK] integer	patient_id text	timeslot_id integer	preference integer	score integer	status integer
1	2	26057784758	1251	1	37	0

```
request(2)
patient_id(2        84758)
timeslot_id(2,1251)
preference(2,1)
score(2,37)
```

Figure 4: Database record (a), conversion in merged encoding (b), conversion in split encoding(c)

To resolve these issues different forms of encoding were considered for the optimization process. In the end, using a split approach in the process of translating the database records to knowledge base predicates two significant advantages emerged. The first pertains to the linguistic aspect of ASP as it results in the declarative rules of the ASP program to resemble natural language sentences. This is achieved by splitting each table's attribute and directly translating it to a predicate. This means that for every attribute in the database a new class is created containing in the case of a primary key, only the key's value and for all the other attributes of a table the record's primary key and the attribute's value. Although this approach greatly simplifies the process of developing the clingo code and increases the code's readability it also increases the internal complexity of the knowledge base slowing down the basic database transactions (select, insert, update, delete) thus making it less suitable for the data management inside the knowledge base. For example, in order to resolve a select query to a split encoded knowledge base it is necessary to first select all the primary keys that conform to the given conditions and then collect all the needed attributes matching the obtained primary keys. It also produces a knowledge base with more predicates leading to a larger data structure for the solver to handle.

Optimal rescheduling with ASP

Typically, every ASP program starts with the creation of its knowledge base which functions as its input [10]. Using the knowledge base, auxiliary predicates required to express facts and relations necessary for the problem's description are created. At this point the core rules for the generation of all possible answer sets are stated. These answer sets, in their current form, are not always either correct (able to satisfy the real problem) or optimal. They just describe a potential solution of the problem's state space. To reach a solution that satisfies the real problem, integrity constraints are applied, that is to say, each potential solution is tested according to the logical constraints imposed by the real world, becoming an actual solution if it complies.

As each answer set describes a possible set of actions the set generation rules are often referred to as effect axioms. On the contrary, rules that specify immutable subsets of the answer sets are labeled as frame axioms. Finally, an aggregate rule is often used to specify the parameters and the type of the optimization process. In our case a

Interface structure

Input: Database name, schema, credentials, and conditions
Solver parameters: Path to solver, Output predicates, Time and model constraints, Input data subset and the type of encoding as a boolean
Output: The optimized answer set and optimization statistics

case

b = KnowledgeBase (name, schema, credentials, conditions) # Create the knowledge base
2: kb.toFile() (optional) # Export the knowledge base as a text file for a better data overview
3: Create output auxiliary predicates to express the solution
4: solution = kb.run (path, outPreds, searchDuration, models, subKB, merged) # Run an ASP program on the KB data
5: Process the solution data (optional)
':b.update(entity, conditions, values) # Update KB and with the results of the optimization process

Figure 5: Basic documentation for the Knowledge Base class

maximization command was given taking into consideration the score of all the granted requests. What happens at a technical level is that the solver sums the parameter specified for every actual solution. If the user wants to print one or more predicates from an answer set, a show command can also be used.

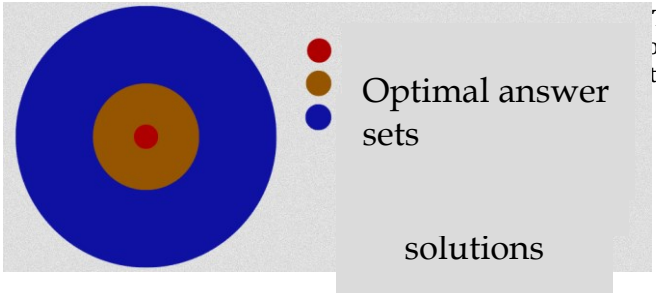


Figure 6: Venn diagram describing the ASP development process.

The clingo rescheduler was developed according to the ASP model described above. Independently of the approach used to encode the knowledge base (split or merged), six auxiliary predicates are used to fully describe the initial schedule.

1. The **granted** predicate that is satisfied by all requests that are granted thus having a state value equal to one.
2. The **appointed** predicate that is satisfied by all the patients having a granted request.
3. The **best** predicate that is satisfied by the single highest scoring request among all the patient's requests. It is generated by arguing that there is no other request belonging to the specific patient that has a greater score.

```
:- grant(R1), timeslot_id(R1, T), grant(R2), timeslot_id(R2, T), R1 != R2.
```

4. The **onlyOption** predicate that is satisfied by the request belonging to a patient that has made one and only request. It is generated by arguing that there is no other request belonging to the specific patient in general.
5. The **singleRequest** predicate that is satisfied a request that is the first and only request in a timeslot's queue. It is generated by stating that there is no other request targeting this timeslot.
6. The **bestSingleRequest** predicate that is satisfied by the patient's best (highest scoring) request from those requests satisfying the singleRequest property. It is generated by arguing that there is no other single request belonging to this patient with a higher score.

The generated answer sets cover the owing three possible types of requests eligible the system to grant:

1. The **granted** but **not best** requests implying that a patient can retain their currently owned appointment if it appears in the answer set that maximizes the common benefit.
2. The **not granted** requests of a patient already having a **granted** request, that present a **higher score** than the one **currently appointed** to the patient. This enables a patient to receive a request with higher priority if it helps to maximize the common benefit.
3. All the requests that belong to a patient who has not been **appointed** a timeslot if this leads to the optimal answer set.

As for the frame axiom, it generates the **claimed** predicate which is satisfied for every request that is already **granted** and is also the **best** request of the corresponding patient. This practically means that all the already claimed requests cannot be granted to another patient.

At this point, eight **integrity constraints** were applied to the answer sets targeting three specific areas, **logic**, **justice,** and **optimization**.

```
:- request(R,_,T,_,_), grant(R,_), request(X,_,T,_,_), grant(X,_), R != X.
```

For the logic part two constraints were applied.

1. Each timeslot can be appointed to only one patient.
2. Each patient can only receive one timeslot (from a specific specialty if the general scope is used).

Figure 7: The first integrity constraint (only one granted request per timeslot) expressed in clingo for the merged approach (a) and the split approach (b)

The danger that appears when someone approaches this problem with the maximization of the common benefit in mind is that many times the system behaves unfairly towards the individual. To prevent this possibility and thus create a just system the following two constraints were applied.

1. If a patient had an appointment in the previous schedule a timeslot must also be granted to that patient after the rescheduling.
2. If a request is a patient's only option, it cannot be dismissed for the sake of a lower scoring request even if it leads to a chain reaction that maximizes common benefit. If we don't apply this constraint a patient with only one request will most probably never receive an appointment.

The only acceptable case in which the individual gain will be sacrificed is when a better option becomes available for an already appointed patient and if granted will lead to a sub optimal common benefit. As can be seen in the example given in Section 2 George's best request was not granted to him because he already had an appointment, resulting in the maximum common benefit without negatively impacting him.

As for the optimization component, in order to reduce the execution time of the clingo rescheduler four presuppositions were introduced to the system reducing the number of requests to be examined for granting in the pursuit of the optimal answer set.

1. All requests that claim an already claimed timeslot will not be taken into consideration.
2. All the single requests that are not the patient's best single request will not be taken into consideration.
3. If a request is a patient's best single request and it has a better score than the one already granted to the patient it will automatically be assigned to the patient blocking all the other requests made by the patient.
4. If a single request is attributed to an unappointed patient, the system should grant it automatically and block all the other requests made by the patient. Both rules 3 and 4 intend to maximize the number of patients served by the system.

Finally, a maximization rule is given to the solver leading to the selection of the answer set that satisfies all the constraints and simultaneously offers the highest common benefit through the grant action.

Evaluation

To assess whether the solution presented in this paper is a viable option for a real healthcare system a data fabricator class was developed in Python to produce pseudo-realistic data according to the database design described in subsection 3.1. The assumption made to produce the timeslot matches most conventional healthcare systems by assigning eight one-hour appointments to each doctor and provides the patients with the freedom to choose as many timeslots as they please with an average of two requests per patient. Also, to be more modular all the parameters needed to create the healthcare system such as the number of patients, the number of doctors, the time period to be simulated (in days), the timeslot availability and the average demand the healthcare system has to face can be tuned by the user. In this case, a timespan of two weeks (10 working days) was assumed for all the datasets. The tests were first run using a version of the rescheduler which assumes that a patient can request a timeslot from any medical department and is able to only receive one at the time of the rescheduler's execution. This assumption was made to test the rescheduler in a context of a general scope where the AI has knowledge of the whole healthcare system. This does not necessarily have to be the case as described in subsection 3.3 where a solution which also takes specialties into consideration is presented. The per-specialty approach was used to test the performance capabilities of the rescheduler in a very high demand scenario where there is a multitude of requests targeting every timeslot offered by a medical department. In reality, the requests would be more evenly distributed among the medical departments. Finally, the option of creating the schedule described in Section 2 intended for validity check purposes is also provided.

Table 3
Execution time for three different datasets using split encoding

Timeslots	Patients	Requests	Execution Time (s)	
			Best	Worst
400	500	1000	0.06	2.79
960	1200	2400	0.18	152.07
2000	2500	5000	421.4	>1800

After extensive testing with various datasets, it seems that the only intrinsic variable meaningfully affecting the execution time of the rescheduler is the number of timeslots offered by the system. An outside factor that also seems to greatly affect the execution time is the prior state of the schedule. If the previous scheduling granted for example the lowest scoring request for every timeslot the rescheduling time appeared to be multiple times higher than in the case of a more logical scheduling that had previously granted the highest scoring requests for every timeslot. For that reason, both scenarios have been measured and assuming a reasonable demand of 2.5 (the requests being 2.5 times the number of timeslots) the best and worst execution times shown in Tables 3 and 4 were obtained. Two other factors observed to affect the execution time to a smaller degree are the patients to requests ratio and the requests to timeslots ratio (demand). In the first case an

increase in the ratio leads to a greater execution time since the effect of the integrity constraint that allows each patient to only one appointment decreases. In the second case an increase in the ratio also leads to a greater execution time since the effect of the integrity constraint that allows each timeslot to be granted to only one request decreases. In every case the introduction of variation in the model makes the optimization process more computationally demanding. Moreover, the distribution of requests over the timeslots and patients over requests can affect the execution time. Namely, a more even distribution will lead to more effective combinations increasing the number of possible answer sets to be examined at the optimization stage. Finally, the type of encoding does not affect the result of the optimization. This was expected due to the identical logic statements used to express the problem. Unlike other attempts on ASP scheduling [12] where the difference in execution time between different encodings can be attributed to the difference in the rule content of the ASP program, differences in the execution time for this implementation based on the encoding of the database for two logically identical reschedulers can be caused by the difference in the optimization path followed by the solver. This can manifest in the case where multiple answer sets provide the same maximum community benefit. To be more precise, according to the syntax of the rules and the expression of the predicates clingo might choose a different path to the optimal answer set or reach a different optimal answer set altogether. In terms of performance the two encodings perform similarly for a small number of models. When the number of models becomes higher (above 200) the cost of choosing a longer path to the optimal answer set in terms of execution time becomes more noticeable. The execution times for three different datasets are displayed in tables 3 and 4 for the merged and split encoding respectively. From the above, we can also conclude that the number of predicates in a knowledge base does not affect the execution time of the rescheduler to a noticeable degree.

Table 4
Execution time for three different datasets using merged encoding

Timeslots	Patients	Requests	Execution Time (s)	
			Best	Worst
400	500	1000	0.09	4.19
960	1200	2400	0.15	448.9
2000	2500	5000	516.9	> 1800

At this point, it can also be useful to grasp the scale of the test data and the complexity faced by the rescheduler. Given a scenario of T timeslots, R requests and d demand with a relatively even distribution the possible combinations are 2^R or $2^{d \cdot T}$ since every request can have only two states, granted, or not granted. This leads, even for a relatively small number of timeslots, to an extremely large number of possible combinations. Although this number is greatly reduced by the application of integrity constraints the fact remains that the complexity needed to be handled by the rescheduler is significant to say the least. As for a real-life application, in order to offer 2000 timeslots in two weeks with the assumptions previously mentioned, a hospital has to employ 25 doctors in a certain specialty, a realistic number for outpatient clinics in Greece.

Challenges of real-world implementation

To implement the rescheduler in the real world one must consider how the most basic assumption previously made will be realized. It is unreasonable to bluntly assume that given a better option at any point in time a patient will always choose to reschedule their appointment. Thus, for every request that was not granted and now satisfies the grant predicate the system will have to ask the patient for permission to change the appointment. One of the points where ASP solvers shine is that they not only give the optimized solution but are also able to describe the steps needed to reach it, thus having the ability to reason about the optimization process. Using this feature, for every time a granted request is canceled, the rescheduler will be executed, producing as its output an action chain that can lead to the schedule providing the higher community benefit. Following this chain, the system can notify the patients with an automated message giving the opportunity to claim their better option. At this point a time limit must be set for the patients to accept or reject their new option. Once a patient with a prior place in the chain accepts a request of higher preference than the one currently owned, the next patient will be notified continuing the confirmation process of the action chain in the case of a positive answer. Thus, each chain produced by clingo will only be realized up to the request prior to the one where the inability to re-schedule the appointment due to a lack of confirmation arises for the first time, resulting in the breaking of the chain. Only after a patient has confirmed the appointment change will the database be updated. This means that if a confirmation process is ongoing and a new cancelation takes place, the following options appear. The first option is to wait for the ongoing process of validating the previous rescheduling action chain to finish and then run the rescheduler with the new state of the database as its input. Another and probably most realistic approach is to break the confirmation process of the previous action chain and run the rescheduler after updating the state of the database using only its currently con-firmed part. In no case can the execution of the rescheduler precede or coincide with the confirmation process as this can lead to fueling the rescheduler with an obsolete input that corresponds to a state of the database where

rescheduling actions caused by the last cancelation have not been taken into consideration.

The second aspect of this solution that needs to be addressed is the practical limitations of its execution time. It was observed that even in the best-case scenario after assuming a logical scheduling has preceded the rescheduling, if the number of timeslots exceeds 3000 or the distribution of data is extremely unfavorable, the time needed for the solver to output the optimal answer set renders it almost unusable in real circumstances. To overcome this challenge three actions can be taken. The first is to simply keep optimizing the code by adding rules that reduce the number of actual solutions taken into consideration during the optimization stage. Although the most theoretically correct approach, even if it was possible to forever continue optimizing, eventually each new optimization will have diminishing returns. To overcome this limitation, as is the case with most serial programs, a restructuring of the rescheduler's code to run in parallel [2] will most probably cause a significant decrease in execution time. Another more practical approach is to set a time limit for the optimization process, interrupting it and retrieving the best available model when it expires. Following that, the system can compare the current community benefit to the newly acquired by the solver and choose the highest scoring one between them. This feature has been implemented using an interrupt in the on_model function used by the solver.

The third available solution, which does sacrifice the general optimality of the output but at the same time grants great flexibility to the application, is the use of a batching approach. Choosing to run the rescheduler on a subset of the data is probably the most useful method to reduce execution time for every kind of rescheduler even more so in a scenario like the medical appointment rescheduling where clear divisions can be made to separate the data needed for optimization according to each medical specialty (as was used in Section 4 for testing) or each doctor. This approach is really an attempt to scale down the problem to a department or an individual level, making the timespan and the demand able to be handled by the rescheduler much wider. The optimality tradeoff increases with each specialization step, and it lies with the system's designer to find the right balance between performance and general maximization of the community benefit. It is worthwhile to mention that the last layer of specialization, providing the most localized maximization of community benefit, will always lie to the resource in demand in this case the doctor's time, a surgical room in ORS [13], a nurse's shift in nurse rescheduling [14] or a meeting room for a business focused application.

Conclusion

The capabilities of ASP solvers to optimize problems with an otherwise unmanageably large search space has yet to be utilized widely by the software industry. One of the obstacles has always been the lack of interfaces between the fundamental software components and the solvers. In this paper a software solution to bridge the gap between the database technologies and the clingo scheduler on a high-level programming language was presented. This solution developed in Python was used for the development of a fully functional application to manage large scale medical appointment scheduling and rescheduling. A theoretical analysis of the database design, interface structure and clingo code was made to aid the reader in understanding the logic used to solve the problem with a new perspective, the community benefit maximization approach, thus showing how the new software capabilities provided by integrating AI to modern software projects can revolutionize the service sector making it [7]fair and optimized. During the evaluation process, the application was thoroughly tested with the results being very promising especially when a 'best request granted' scheduling precedes the application's execution. Even without the scheduling stage the solver proved capable of handling a significant number of patients. It is certainly the case that further optimization and improvements can be made both in the clingo and in the Python fronts to make the application more user-friendly, increase performance and provide new features enabling the rescheduler to handle real-life scenarios faced by large organizations such as hospitals and companies with large scale meeting scheduling requirements. As with many other ASP programs, this application is very adaptable to every sector in need of scheduling and rescheduling solutions, thus making the study of this paper beneficial for both understanding the ASP development process and for providing a base for future applications wanting to solve similar problems.

References

[1] V. Lifschitz, Answer Set Programming, Texas: Springer, 2019.

[2] H. Hoos, R. Kaminski, M. Lindauer and T. Schaub, "aspeed: Solver Scheduling via Answer Set," 2013.

[3] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri and F. Scarcello, "The DLV System for Knowledge Representation," 2003.

[4] M. Hawkins, "Survey of Physician Appointment Wait Times and Medicare and Medicaid Acceptance Rates," AMN Healthcare, 2022.

[5] A. H. Gertz, C. C.Pollack, M. D.Schultheiss and J. S. Brownstein, "Delayed medical care and underlying health in the United States during the COVID-19 pandemic: A cross-sectional

study," *Preventive Medicine Reports,* 2022.

[6] J. Gier, "Healthcare Innovation," SCI Solutions, [Online]. Available: https://www.hcinnovationgroup.com/clinical-it/article/13008175/missed-appointments-cost-the-us-healthcare-system-150b-each-year. [Accessed August 2023].

[7] N. Jukić, S. Vrbsky, S. Nestorov and A. Sharma, "ERD Plus," 2015. [Online]. Available: https://erdplus.com/. [Accessed 2023].

[8] J. Perez, "DB Designer," 2006. [Online]. Available: https://erd.dbdesigner.net/. [Accessed 2023].

[9] M. Krötzsch, "Knowledge Graphs - Lecture 8: Limits of SPARQL," slide 17, 15th Dec 2020.

[10] G. Brewka, T. Eiter and M. Truszczyński, "Answer Set Programming at a Glance," 2011.

[11] O. E. Khatib, "Job shop Scheduling under Answer Set Programming Environment," *International Journal of Engineering and Innovative Technology (IJEIT),* vol. 5, no. 5, 2015.

[12] C. Dodaro and M. Maratea, "Nurse Scheduling via Answer Set Programming," 2017.

[13] C. Dorado, G. Galatà, M. Kamran, M. Maratea and I. Porro, "Operating Room (Re)Scheduling with Bed," 2021.

[14] M. Alviano, C. Dodaro and M. Maratea, "Nurse (Re)scheduling Via Answer Set Programming," 2014.