National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

Multi-objective mobile robot path planning applied in autonomous robotic inspection of outdoor environment

Diploma Thesis

of

ANDREAS VATISTAS

Supervisor: Costas Tzafestas
Assoc. Prof. NTUA

Athens, March 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

Multi-objective mobile robot path planning applied in autonomous robotic inspection of outdoor environment

Diploma Thesis

of

ANDREAS VATISTAS

Supervisor: Costas Tzafestas
Assoc. Prof. NTUA

Approved by the examination committee on 28th March 2024.

(Signature)	*(Signature)*	*(Signature)*
....	....	....
Costas Tzafestas	Athanasios Rontogiannis	Haralambos Psilakis
Assoc. Prof. NTUA	Assoc. Prof. NTUA	Lect. NTUA

Athens, March 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

(Signature)

. . . .
Andreas Vatistas
28th March 2024

Abstract

In this thesis we propose a solution to a multi-objective optimization problem for an autonomous inspection task conducted in an outdoor environment, using a mobile robot. A user selects certain elements that require inspection, and our programs based on this input and the environment's map information suggest the optimal sequence of waypoints to be visited. The metrics we account for are both the time constraint of the total movement, directly related to the total distance and energy consumed, and the maximization of the visual and infrared data captured during the inspection. We thoroughly analyzed related work found in the literature, proposed and implemented a methodology to address the aforementioned challenge. To achieve this we transform the whole problem into smaller ones and solve each one separately. This, firstly, involves an interesting point selection task and a path planning one, both of which take place offline as pre-processing steps. Then based on these data and the user's input, we transform the extracted problem into a Generalized Traveling Salesman Problem, and employ the GLNS Solver to solve it, as it demonstrates superior overall performance compared to the other state of the art approaches.

Keywords

Mobile robotics, Path planning, Autonomous inspection, Multi-objective optimization, Generalized Traveling Salesman Problem (GTSP)

Περίληψη

Σε αυτή τη διπλωματική προτείνουμε μία λύση για ένα πρόβλημα βελτιστοποίησης πολλαπλών κριτηρίων που αναφέρεται σε μία εργασία επιθεώρησης στοιχείων εξωτερικού χώρου με χρήση αυτόνομου κινούμενου τετράτροχου ρομπότ. Ένας χρήστης επιλέγει συγκεκριμένα στοιχεία που κρίνει χρήσιμο να επιθεωρηθούν, και τα προγράμματά μας με τη σειρά τους βάσει της επιλογής αυτής και πληροφοριών για το χάρτη του περιβάλλοντος αυτού, που είναι από πριν γνωστές, προτείνουν τη βέλτιστη ακολουθία από σημεία που πρέπει να επισκεφτεί το ρομπότ για να φέρει σε πέρας με βέλτιστο τρόπο την προαναφερθείσα εργασία. Οι μετρικές που λαμβάνουμε υπόψιν είναι τόσο ο συνολικός χρόνος της κίνησης, που έχει άμεση συσχέτιση με τη συνολική απόσταση και ενέργεια που θα καταναλωθεί τελικώς, καθώς και η συνολική οπτική και υπέρυθρη πληροφορία που θα συλλεχθεί. Αναλύσαμε διεξοδικά σχετικά έργα της βιβλιογραφίας και σχεδιάσαμε και υλοποιήσαμε μία μέθοδο για το σκοπό αυτό. Σύμφωνα με αυτή, μετατρέπουμε το πρόβλημα σε μικρότερα και λύνουμε το κάθε ένα ξεχωριστά. Αυτά αποτελούνται αρχικά, από μία διαδικασία εύρεσης πιθανών σημείων ενδιαφέροντος που αποτελούν καλές εικονοληπτικές θέσεις και μία διεργασία σχεδίασης μονοπατιών, τα οποία διενεργούνται προπαρασκευαστικά για τη συνέχεια. Ακολούθως, βάσει αυτών των δεδομένων και της επιλογής του χρήστη, μετατρέπουμε το πρόβλημα σε Γενικευμένο Πρόβλημα Πλανόδιου Πωλητή και το λύνουμε χρησιμοποιώντας τον GLNS Solver, που παρουσιάζει τις καλύτερες επιδόσεις συγκριτικά με τους υπόλοιπους state of the art λύτες.

Λέξεις Κλειδιά

Κινητά ρομπότ, Αλγόριθμοι σχεδίασης μονοπατιών, Αυτόνομη επιθεώρηση, Πολυκριτηριακή βελτιστοποίηση, Γενικευμένο Πρόβλημα Πλανόδιου Πωλητή

Acknowledgements

First and foremost, I would like to thank my supervisor Assoc. Prof. Costas Tzafestas for inspiring me with his highly instructive lectures, for his guidance, support, the freedom of thinking and implementing and the opportunity to work and discuss ideas with highly skilled individuals.

I would also like to thank Dr. George Moustris, Ioannis Alamanos and Panagiotis Mermigkas for our fruitful discussions, for their mentoring and for sharing their thoughts, ideas and knowledge in the field.

The completion of this thesis marks the end of my 5-year studies at the School of Electrical and Computer Engineering of National Technical University of Athens (NTUA). At this point, I would really like to express my appreciation to all my professors who have taught me a lot these years and my fellow students who have been part of this experience and contributed to an environment of collaborative learning and growth.

Last but definitely not least, I want to thank my family and friends who always stand by my side in every possible way.

Athens, March 2024

Andreas Vatistas

Contents

List of Figures

Εκτεταμένη Περίληψη

1. Εισαγωγή

Ως κινητά ρομπότ ορίζονται τα ρομπότ που δεν έχουν σταθερή βάση και έχουν τη δυνατότητα να μετακινούνται ελεύθερα σε διάφορα περιβάλλοντα.

Πρόβλημα βελτιστοποίησης λέγεται ένα πρόβλημα για το οποίο κανείς καλείται να δώσει την καλύτερη λύση μεταξύ πολλών πιθανών λύσεων.

Η εργασία μας αναφέρεται σε ένα μικρό αυτόνομο τετράτροχο κινητό ρομπότ, για το οποίο καλούμαστε να αναπτύξουμε αλγορίθμους για να επιλύσουμε ένα πρόβλημα βελτιστοποίησης αναφορικά με μία εργασία επιθεώρησης σε κέντρα υψηλής τάσης του ΑΔΜΗΕ (Ανεξάρτητος Διαχειριστής Μεταφοράς Ηλεκτρικής Ενέργειας) στα πλαίσια του προγράμματος ΕΝΟΡΑΣΗ. Η διαδικασία και η μέθοδος που προτείνουμε είναι πιο γενική και μπορεί να επεκταθεί προφανώς και σε άλλες εφαρμογές.

Το πρόβλημα έχει ως εξής. Υπάρχουν διάφορα στοιχεία, ηλεκτρολογικά στην παρούσα φάση, τα οποία θέλουμε να επιθεωρήσουμε με αυτόνομο τρόπο στο εξωτερικό αυτό περιβάλλον που κινείται το ρομπότ, όπως φαίνεται και στην Εικόνα 1. Η επιθεώρηση πραγματοποιείται τραβώντας φωτογραφίες σε ορατό φάσμα, αλλά και υπερύθρες. Έχουμε στη διάθεσή μας ένα χάρτη που απεικονίζει την τοπολογική αναπαράσταση του εξωτερικού πεδίου, δείχνει για παράδειγμα την κλίση σε καθε σημείο και άλλες πληροφορίες που καταλήγουν στην κατηγοριοποίηση κάθε κελιού ως προσβάσιμου ή μη. Ο όρος κελί αναφέρεται σε μια περιοχή μικρή γύρω από ένα καρτεσιανό σημείο του χώρου. Επιπλέον, έχουμε στη διάθεσή μας χάρτες που παρουσιάζουν την οπτική πληροφορία, δηλαδή το πόσο καλά βλέπει κάθε κελί τον κάθε πιθανό στόχο.

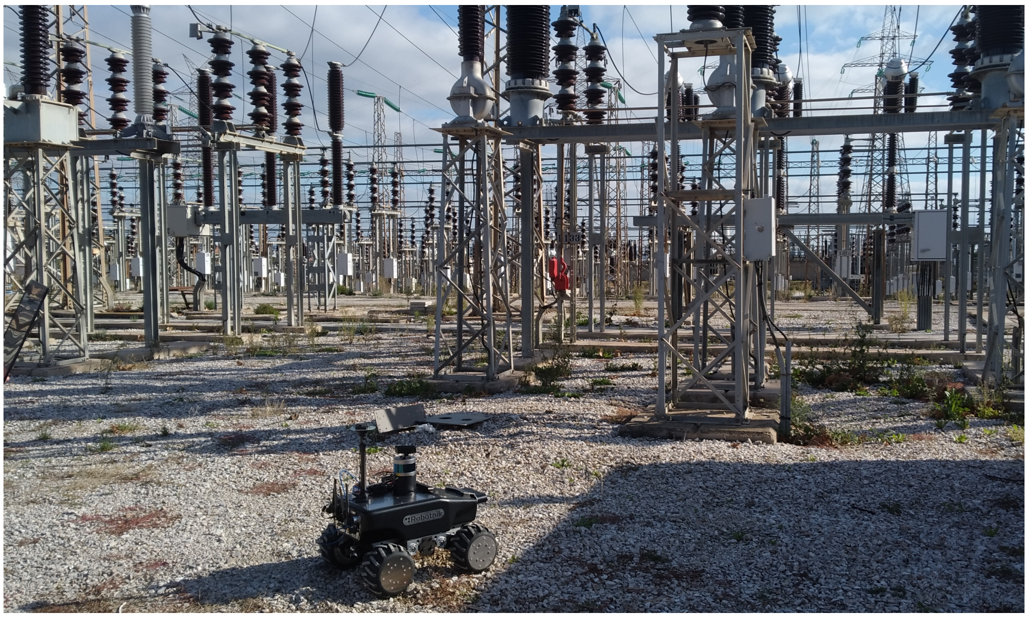


Figure 1. *Το ρομπότ που έχουμε στη διάθεση μας στο εξωτερικό περιβάλλον του Κέντρου Υψηλής Τάσης (ΚΥΤ) της Παλλήνης. Φαίνονται πίσω και μερικά πιθανά στοιχεία προς επιθεώρηση, πάνω στους αντίστοιχος στήλους.*

Ένας χρήστης λοιπόν, θα επιλέγει μερικά στοιχεία τα οποία θεωρεί ότι θέλουν επιθεώρηση. Οι αλγόριθμοί μας καλούνται να προτείνουν τη βέλτιστη αλληλουχία από σημεία στο χώρο αυτό που το ρομπότ θα πρέπει αυτόνομα να ακολουθήσει, προκειμένου να φέρει σε πέρας την αποστολή επιθεώρησης με βέλτιστο τρόπο. Ως βέλτιστο στην παρούσα εφαρμογή, ορίζουμε τον τρόπο που μεγιστοποιεί την οπτική πληροφορία, ενώ συγχρόνως ελαχιστοποιεί το συνολικό κόστος μετάβασης μεταξύ των σημείων αυτών. Προφανώς για την επιθεώρηση κάθε στοιχείου υπάρχουν πολλά πιθανά αξιόλογα σημεία από τα οποία μπορεί να γίνει η επιλογή για τη συλλογή των δεδομένων.

Η μέθοδος που αναπτύξαμε και υλοποιήσαμε χωρίζει το πρόβλημα σε υπο-προβλήματα τα οποία με τη σειρά τους λύνονται με βέλτιστο δυνατό τρόπο. Ενδεικτικά, τα κύρια υπο-προβλήματα της διαδικασίας είναι η εξαγωγή των πιθανών σημείων επίσκεψης για την επιθεώρηση κάθε στοιχείου. Η προεπεξεργασία των δεδομένων με χρήση αλγορίθμων για τον υπολογισμό του κόστους μετάβασης μεταξύ αυτών. Τέλος, βάσει των παραπάνω πληροφοριών και την επιλογή του χρήστη, η εξαγωγή του βέλτιστου πλάνου ως λύση.

Στα επόμενα κεφάλαια παρουσιάζονται κατά σειρά, περιληπτικά αποτελέσματα από τη διεξοδική βιβλιογραφική έρευνα που πραγματοποιήσαμε, η προσέγγισή μας για τη λύση του προβλήματος, καθώς και συμπεράσματα και προτάσεις για μελλοντική έρευνα επί του θέματος.

2. Σχετική Έρευνα

Οι αλγόριθμοι σχεδίασης μονοπατιών (path planning) αποτελούν ένα μεγάλο ερευνητικό κομμάτι στα πλαίσια της ρομποτικής. Ασχολούνται με την εύρεση συντομότερων μονοπατιών μεταξύ ενός αρχικού σημείου και ενός σημείου στόχου σε ένα περιβάλλον. Καλούνται να απαντήσουν, αποφεύγοντας πιθανά εμπόδια και συγχρόνως βελτιστοποιώντας διάφορες μετρικές, όπως ο χρόνος της κίνησης ή τα συνολικά μέτρα, μετρικές ασφάλειας (κίνηση αρκετά μακριά από εμπόδια για παράδειγμα), ή και άλλες μετρικές που σχετίζονται με την εκάστοτε εφαρμογή. Πολλές τεχνικές σε αυτή την κατεύθυνση έχουν αναπτυχθεί. Ενδελεχώς παρουσιάζεται πολύ υλικό στις εργασίες [7, 8, 9, 10]. Όλοι οι συγγραφείς των παραπάνω επισημαίνουν τη δυσκολία των προβλημάτων αυτών και διαπιστώνουν ότι κυρίως εξαρτώνται από παράγοντες όπως οι διαστάσεις του χώρου της εκάστοτε αναζήτησης και περιορισμοί στους υπολογιστικούς μας πόρους. Για τη μεγιστοποίηση των αποτελεσμάτων, γίνονται κάποια trade-offs. Για το σκοπό αυτό, οι μέθοδοι χωρίζονται σε δύο μεγάλες κατηγορίες: online & offline. Οι αλγόριθμοι που αναφέρονται στην πρώτη κατηγορία σχεδιάζουν μονοπάτια ¨ζωντανά¨, την ώρα της κίνησης δηλαδή, οπότε λαμβάνουν υπόψη και δυναμικούς περιορισμούς όπως κινούμενα εμπόδια. Αυτοί που ανήκουν στη δεύτερη κατηγορία σχεδιάζουν τις διαδρομές από πριν, έχουν συνεπώς και περισσότερο χρόνο για διεξοδικότερη ανάλυση, αλλά αναφέρονται σε στατικά περιβάλλοντα.

Το παραπάνω πρόβλημα μπορεί να αντιμετωπιστεί με διάφορους τρόπους, ένας από αυτούς είναι με αλγόριθμους αναζήτησης σε γράφο που έχει προκύψει βάσει της τοπολογικής αναπαράστασης του χώρου στον οποίο αναφερόμαστε [11]. Χρήσιμοι τέτοιο αλγόριθμοι παρουσιάζονται ακολούθως.

Ο αλγόριθμος του Dijkstra, είναι ένας αλγόριθμος αναζήτησης γράφου που επιλύει το πρόβλημα του συντομότερου μονοπατιού για οποιονδήποτε κατευθυνόμενο (ή μη) γράφο με μη-αρνητικά βάρη στις ακμές [12]. Ξεκινά από έναν κόμβο πηγή και εξερευνά επαναληπτικά τους γειτονικούς κόμβους, ενημερώνοντας την συντομότερη απόσταση προς κάθε κόμβο μέχρι να επισκεφθεί όλους τους κόμβους. Μπορεί επίσης να χρησιμοποιηθεί για την εύρεση των συντομότερων μονοπατιών από έναν κόμβο προς έναν μόνο κόμβο προορισμού, διακόπτοντας τον αλγόριθμο μόλις καθοριστεί το συντομότερο μονοπάτι προς τον κόμβο αυτόν.

Ο αλγόριθμος A* είναι ένας αλγόριθμος που χρησιμοποιείται για την αναζήτηση βέλτιστων μονοπατιών και τη διάσχιση γράφων. Ξεκινώντας από έναν αρχικό κόμβο και γνωρίζοντας ποιος είναι ο κόμβος στόχος, σε έναν γράφο με βάρη, επιδιώκει να βρει το βέλτιστο μονοπάτι μεταξύ αυτών. Για το το σκοπό αυτό χρησιμοποιεί έναν ευρετικό μηχανισμό, ευριστική συνάρτηση, που κάθε φορά προβλέπει την απόσταση από τον τρέχοντα κόμβο στον κόμβο στόχο. Ο A* λαμβάνει υπόψη τόσο το πραγματικό κόστος του μονοπατιού από την αρχική κατάσταση ως τον τρέχοντα κόμβο όσο και το εκτιμώμενο κόστος για την επίτευξη του στόχου από αυτόν τον κόμβο, βάσει της ευριστικής συνάρτησης που έχει επιλεχθεί. Οι αποφάσεις λαμβάνονται βάσει του παραπάνω αθροίσματος. Έτσι, ο αλγόριθμος επιλέγει με έξυπνο τρόπο τους κόμβους που πρέπει να εξερευνήσει, οδηγώντας σε ένα βέλτιστο μονοπάτι με μικρότερο τελικό υπολογιστικό φόρτο, καθώς περιορίζεται/κατευθύνεται αρκετά η αναζήτηση. Οι Peter Hart, Nils Nilsson και Bertram Raphael του Stanford Research Institute δημοσίευσαν πρώτοι τον αλγόριθμο το 1968 [13].

Τα προβλήματα που λέμε ότι ανήκουν στην κλάση NP, αποτελούν μία κατηγορία υπολογιστικών προβλημάτων για τα οποία οι λύσεις τους μπορούν να επαληθευτούν σε πολυωνυμικό χρόνο. Ωστόσο, η εύρεση μιας λύσης μπορεί να είναι σημαντικά πιο πολύπλοκη αλγοριθμικά. Έχουν λάβει μεγάλη προσοχή και έχει γίνει αρκετή έρευνα λόγω της ιδιαίτερης φύσης τους.

Ένα από τα πιο γνωστά προβλήματα που ανήκουν στην παραπάνω κλάση, είναι το πρόβλημα, γνωστό και ως **Πρόβλημα του Πλανόδιου Πωλητή**. Το πρόβλημα αυτό θέτει την ακόλουθη ερώτηση: ¨Λαμβάνοντας υπόψη μια λίστα από πόλεις και τις αποστάσεις μεταξύ κάθε ζεύγους αυτών, ποια είναι η συντομότερη, δηλαδή βέλτιστη, διαδρομή που επισκέπτεται κάθε πόλη και τελικά επιστρέφει την πόλη προέλευσης·' Βρίσκει μεγάλη πρακτική εφαρμογή σε διάφορες καταστάσεις, από το σχεδιασμό διαδρομών για πωλητές, μέχρι την κατασκευή υπολογιστικών τσιπς, ακόμα και κρυσταλλογραφία ακτίνων X [14].

Για ένα συμμετρικό τέτοιο πρόβλημα με πόλεις ν το πλήθος, έυκολα αποδεικνύεται ότι υπάρχουν (ν-1)!/2 πιθανές διαδρομές. Οπότε για $\upsilon = 10$, υπάρχουν περισσότερες από 10^{18} διαδρομές, νούμερο απαγορευτικό για εξαντλητικές λύσεις.

Η ανάγκη για ανάπτυξη προσεγγιστικών αλγορίθμων για την επίλυση του είναι προφανής. Ενδεικτικά αναφέρουμε μερικές μεθόδους στην κατεύθυνση αυτήν. Η μέθοδος του *πλησιέστερου γείτονα* χτίζει με πολύ κατανοητό, απλό τρόπο την διαδρομή. Ξεκινώντας από μία τυχαία πόλη, σε κάθε στάδιο γίνεται επιλογή του πλησιέστερου γείτονα που δεν αποτελεί μέλος της διαδρομής, και τελικά επιστρέφει στον κόμβο που ξεκίνησε η διαδικασία. Προφανώς, επιδέχεται μεγάλες βελτιώσεις. Η μέθοδος *2-opt*, ξεκινάει με μία ήδη υπάρχουσα διαδρομή, ανεξάρτητα με το πως χτίστηκε αυτή. Επαναληπτικά αντικαθιστά δύο συνδέσμους με δύο άλλους που οδηγούν σε καλύτερη τελική συνολική διαδρομή. Η διαδικασία τερματίζεται όταν δεν υπάρχει άλλη τέτοια αλλαγή που να οδηγεί σε βελτίωση. Η Εικόνα 2 δείχνει ακριβώς μία τέτοια αλλαγή.

Μία γενίκευση του σκεπτικού που παρουσιάστηκε παραπάνω, οδήγησε τους *Lin-Kernighan* στην ανάπτυξη δικού τους αλγορίθμου, το 1971. Εφαρμόζοντας μια σειρά συνεχών βελτιώσε-

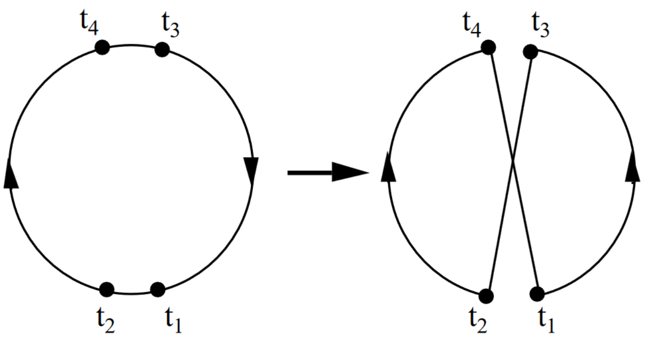


Figure 2. *2-opt κίνηση*

ων στην αρχική λύση που προτείνεται, ο αλγόριθμός τους προσπαθεί να βρει τη βέλτιστη διαδρομή για το πρόβλημα του περιοδεύοντος πωλητή. Για να γίνει αυτό εισάγεται ο ορισμός της $\lambda - opt$ κίνησης. Το σκεπτικό βασίζεται στα εξής [6]: Έστω T ο τρέχων περίπατος. Στην κάθε επανάληψη, ο αλγόριθμος προσπαθεί να βρει δύο σύνολα συνδέσμων, $X = \{x_1, ..., x_r\}$ και $Y = \{y_1, ..., y_r\}$, τέτοια ώστε, αν οι σύνδεσμοι του X διαγραφούν από το T και αντικατασταθούν από τους συνδέσμους του Υ, το αποτέλεσμα είναι ένας καλύτερος περίπατος. Παράδειγμα μίας $3 - opt$ κίνησης παρουσιάζεται στην Εικόνα 3. Πολλά κριτήρια παρουσιάζονται στην εργασία τους για το ποιοι σύνδεσμοι μπορούν να ανήκουν στο κάθε σύνολο. Για παράδειγμα, πρέπει οι σύνδεμοι x_i, y_i να μοιράζονται το ένα άκρο τους, τα σύνολα να μην έχουν κοινά στοιχεία και άλλα παρόμοια (παρουσιάζονται συγκεντρωτικά στο κεφάλαιο 2.2). Επίσης αναφέρθηκαν σε κριτήρια για τη βελτίωση αποδοτικότητας, χρησιμοποιώντας ευρετικούς μηχανισμούς, όπως ο περιορισμός της αναζήτησης στους 5 κοντινότερους γείτονες του κάθε κόμβου, ή ο τερματισμός της όλης διαδικασίας αν ο νέος περίπατος έχει ίδιο κόστος με τον προηγούμενο. Ο Helsgaun έφερε μερικές βελτιώσεις στον παραπάνω αλγόριθμο, βελτιώνοντας τις παραπάνω ευρυστικές, που οδήγησαν στον αλγόριθμο LKH. Εισάγοντας την α-μετρική, οδήγησε σε καλύτερα σύνολα πιθανών γειτόνων βελτιώνοντας αρκετά το κόνσεπτ της ευρετικής γειτνίασης. Αναλυτική περιγραφή αυτών γίνεται στο κεφάλαιο 2.2.

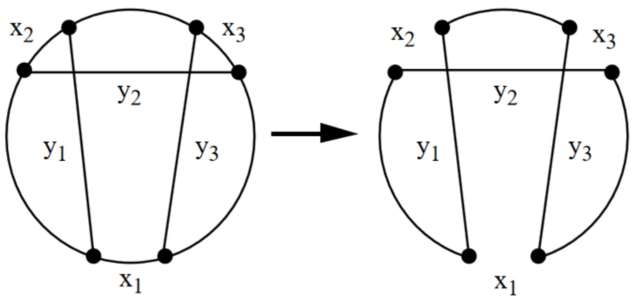


Figure 3. *3-opt κίνηση*

Μία άλλη πολύ γνωστή μέθοδος επίλυσης του Προβλήματος Πλανόδιου Πωλητή είναι αυτή της *Προσομοιωμένης Ανόπτησης*. Ο αλγόριθμος παρουσιάζεται αμέσως παρακάτω. Ξεκινάει από ένα προτεινόμενο τουρ/μονοπάτι σαν λύση και όπως και πριν διαδοχικά το βελτιώνει βασιζόμενος στο $\lambda - opt$ σκεπτικό. Η διαφορά έγγυται στο γεγονός ότι αποδέχεται και λύσεις χειρότερες λύσεις σε κάποιες επαναλήψεις βάσει της πιθανότητας $\exp\left(\frac{f(i)-f(j)}{c_k}\right)$, η οποία βασίζεται στη προσομοίωση της ανόπτησης ενός υλικού. Η παράμετρος c_k που παρουσιάζεται, αναφέρεται στη θερμοκρασία του υλικού, ενώ η συνάρτηση f που μετράει το πόσο καλή είναι η εκάστοτε λύση.

Algorithm 1: *Simulated Annealing [15]*

```
procedure SimulatedAnnealing($i_{start}$, $c_0$, $L_0$)
    INITIALIZE($i_{start}$, $c_0$, $L_0$);
    $k \leftarrow 0$;
    $i \leftarrow i_{start}$;
    while stopcriterion = FALSE do
        for $l \leftarrow 1$ to $L_k$ do
            GENERATE($j$ from $S_l$);
            if $f(j) \leq f(i)$ then
                $i \leftarrow j$;
            else
                if exp $\left(\frac{f(i)-f(j)}{c_k}\right) > \text{random}[0,1)$ then
                    $i \leftarrow j$;
                end if
            end if
        end for
        $k \leftarrow k+1$;
        CALCULATE_LENGTH($L_k$);
        CALCULATE_CONTROL($c_k$);
    end while
end procedure
```

Η **Γενίκευση του Προβλήματος Πλανόδιου Πωλητή (GTSP)** είναι μία επέκταση του βασικού προβλήματος που συζητήθηκε πριν. Βρίσκει μεγάλη πρακτική εφαρμογή και υπάρχει αρκετή βιβλιογραφία λόγω της ιδιαίτερης φύσης και δυσκολίας που παρουσιάζει. Σε αυτό το πρόβλημα το σύνολο των κόμβων χωρίζεται σε γκρουπς/συστάδες. Η ερώτηση που θέτει το πρόβλημα είναι, αντίστοιχα με πριν, ¨ποια είναι η βέλτιστη, μικρότερου συνολικού κόστους διαδρομή που πρέπει να ακολουθηθεί ώστε να επισκεφτεί κανείς τουλάχιστον ένα κόμβο από κάθε γκρουπ από τα προαναφερθέντα¨. Η εκδοχή αυτού του προβλήματος την επίσκεψη για ακριβώς μίας φοράς σε κάθε γκρουπ, αναφέρεται ως Equality Generalized Traveling Salesman Problem (E-GTSP).

Στο σημείο αυτό έγινε ενδελεχής βιβλιογραφική έρευνα, σχετικά με τις διάφορετικες υπάρχουσες προσεγγίσεις επίλυσης, καθώς και ποιες από αυτες έχουν τις καλύτερες επιδόσεις. Υπάρχουν πολλές προσεγγίσεις, ενδεικτικά αναφέρονται οι ακριβείς αλγόριθμοι-exact algorithms, οι οποίοι όμως παρουσιάζουν μεγάλο υπολογιστικό κόστος, μέθοδοι μετασχηματισμού του προβλήματος, αλγόριθμοι μείωσης, γνωστοί και ως reduction algorithms, προσεγγιστικοί αλγόριθμοι αλλά και ευρετικοί, που κάνουν χρήση ευριστικών μεθόδων και παρουσιάζουν τα καλύτερα συνολικά αποτελέσματα.

Έχουν δημιουργηθεί διάφορα σύνολα δεδομένων, παράλληλα με την ανάπτυξη της έρευνας στον τομέα αυτό, τα οποία αποσκοπούν για την αξιολόγηση των διάφορων λύσεων. Αρχικά, η βιβλιοθήκη με δεδομένα GTSP_LIB, η οποία χτίζει τα γκρουπς βάσει της εγγύτητας

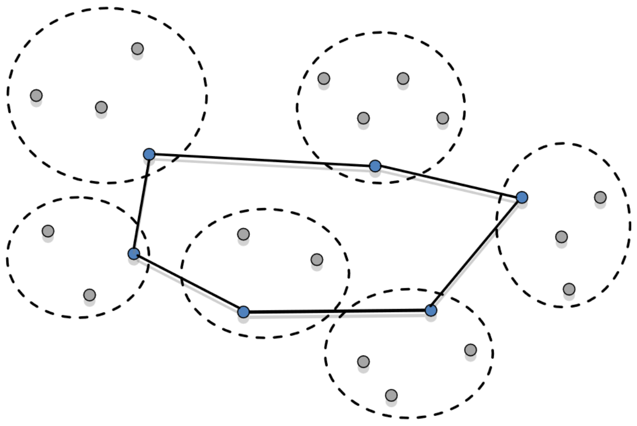


Figure 4. *GTSP παράδειγμα για 6 γκρουπς και 23 συνολικά κόμβους [1]*

των κόμβων. Διαλέγει $m = \lceil n/5 \rceil$ κέντρα, όπου n ο συνολικός αριθμός των κόμβων, και επαναληπτικά ομαδοποιεί τους υπόλοιπους κόμβους στα γκρουπς του πλησιέστερου κέντρου. Άλλα τέτοια σύνολα δεδομένων παρουσιάζονται στις βιβλιοθήκες BAF_LIB, MOM_LIB και LARGE_LIB.

Μία συγκριτική αξιολόγηση των state of the art μεθόδων παρουσιάζεται στην εργασία [2]. Οι τέσσερις καλύτερες μέθοδοι την παρούσα στιγμή στη βιβλιογραφία φαίνεται πως είναι:

- Ο μιμητικός αλγόριθμος που παρουσιάζεται στην εργασία των Gutin & Karapetyan [16],
- Η λύση βασισμένη στον Lin-Kernighan-Helsgaun (LKH) αλγόριθμο, γνωστή και ως GLKH [1],
- Η λύση ¨αναζήτησης μεγάλης γειτονιάς¨, γνωστή και ως G-LNS [17] και
- Η λύση επαναλαμβανόμενης τοπικής αναζήτησης (basic iterated local search) [18]

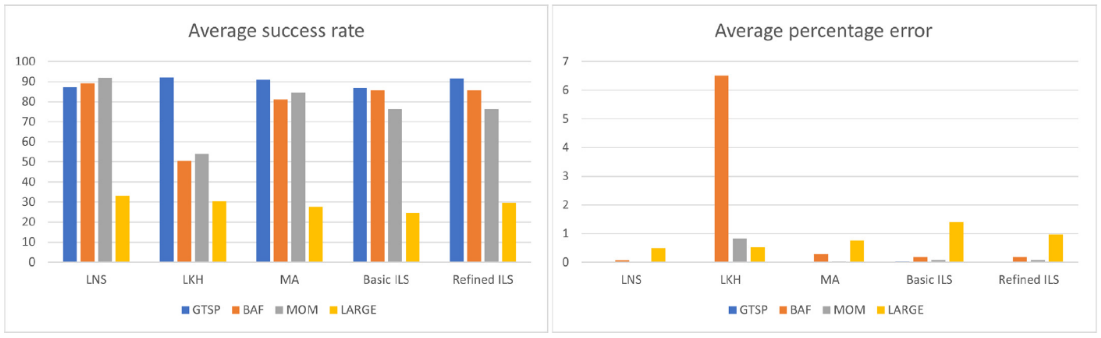


Figure 5. *Συνοπτικά αποτελέσματα της σύγκρισης των καλύτερων μεθόδων [2]*

Τα αποτελέσματα της σύγκρισής τους παρουσιάζονται αρκετά συνοπτικά στα γραφήματα της Εικόνας 5. Τα συμπεράσματα αυτής της ανάλυσης είναι πως η GLKH λύση παρουσιάζει τα καλύτερα αποτελέσματα για GTSP_LIB, ενώ ακολουθεί ελαφρώς πίσω της η GLNS, η οποία δε, στα υπόλοιπα σύνολα δεδομένων παρουσιάζει συντριπτική υπεροχή μεταξύ και των τεσσάρων.

Ο GLKH solver, μετατρέπει το γενικευμένο πρόβλημα πλανόδιου πωλητή, στο κλασσικό απλούστερο πρόβλημα πλανόδιου πωλητή, και το επιλύει με τη μέθοδο LKH που περιγράφηκε προηγουμένως. Για να πραγματοποίησει τη μετατροπή αυτή βασίζεται στο έργο [19], το οποίο προτείνει τα εξής. Έστω V' και c' το νέο σύνολο κορυφών και τα νέα κόστη για το γράφο αντίστοιχα. Ο αλγόριθμος της μετατροπής περιγράφεται ως εξής:

- Το V' ταυτίζεται με το V.
- Μεταξύ κόμβων ίδιας συστάδας δημιουργεί έναν αυθαίρετο κατευθυνόμενο κύκλο με κόστη $c'_{ij} = 0$, για κάθε v_i, v_j, όπου ο κόμβος v_j διαδέχεται τον v_i στον προαναφερθέντα κύκλο.
- Εάν v_i, v_j ανήκουν σε διαφορετικές συστάδες/γκρουπς, ορίζει $c'_{ij} = c_{kj} + M$, όπου v_k ο κόμβος που διαδέχεται τον v_i στον κύκλο , και M μία μεγάλη σταθερά. Αρκεί η τιμή του M να είναι μεγαλύτερη από το άθροισμα των n μεγαλύτερων κόστων μετάβασης.
- Αλλιώς, ορίζει $c'_{ij} = 2M$.

Η μετατροπή αυτή λειτουργεί καθώς το βέλτιστο μονοπάτι για τη λύση του TSP προβλήματος, μόλις συναντήσει έναν κόμβο κάποιας συστάδας, αμέσως επισκέπτεται και όλους τους υπόλοιπους, προτού επισκεφθεί επόμενο γκρουπ από σημεία. Η μετατροπή αυτή δεν είναι κατάλληλη για όλους τους διαθέσιμους λύτες, παρ' όλα αυτά ο LKH μπορεί να εφαρμοστεί χωρίς προβλήματα όπως αποδεικνύεται στο [1].

Ο GLNS solver, που βάσει της προηγούμενης ανάλυσης παρουσιάζει τα καλύτερα συνολικά αποτελέσματα, παρουσιάστηκε το 2017 από τους Smith & Imeson [17]. Λειτουργεί γύρω από το σκεπτικό του adaptive large neighborhood search (ALNS), στα ελληνικά προσαρμοστική αναζήτηση μεγάλης γειτονιάς. Ξεκινάει όπως και πριν με μία προτεινόμενη λύση και επαναληπτικά αφαιρεί και προσθέτει συνδέσμους μέσω ευριστικών συναρτήσεων μέχρι να φτάσει σε μία συνθήκη τερματισμού. Κάποια βάρη σχετίζονται με αυτές τις ευριστικές εισαγωγής και διαγραφής, γι' αυτό και όνομα προσαρμοστική αναζήτηση. Περισσότερες πληροφορίες γι' αυτές μπορεί κανείς να δει στο αντίστοιχο κεφάλαιο της εργασίας μας, και ακόμα πιο αναλυτικά στην αρχική εργασία [17] Ψευδοκώδικας για τον αλγόριθμο ακολουθεί.

Η επιλογή του αριθμού των κορυφών που θα αφαιρεθούν είναι τυχαία, και προκύπτει ομοιόμορφα από το εύρος 1 έως N_{max}. Βασισμένο σε ένα κριτήριο προσομοιωμένης ανόπτησης (γραμμή 16 του αλγορίθμου) αποδέχεται ή απορρίπτει την τροποποιημένη διαδρομή T_{new}. Αρχικό κριτήριο τερματισμού αποτελεί η μη βελτίωση της λύσης μετά από ορισμένες επαναλήψεις του αλγορίθμου. Εν συνεχεία, μερικές επανεκιννήσεις της διαδικασίας πραγματοποιύνται, με την καλύτερη κάθε στιγμή λύση, με μικρότερη θερμοκρασίες προσομοίωσης ανόπτησης όμως. Μετά από μερικά βήματα μη βελτίωσης της λύσης, πάλι, σταματάει και αυτή η διαδικασία.

3. Η συνεισφορά μας

Για την επίλυση του προβλήματος βελτιστοποίησης πολλαπλών κριτηρίων που καλούμαστε να επιλύσουμε, και το οποίο παρουσιάστηκε προηγουμένως, προτείνουμε το χωρισμό του

Algorithm 2: *GLNS(G, P_V) [17]*

```
1:  Input: A GTSP instance $(G, P_V)$.   ▷ G: Graph, $P_V$: partition of vertices V into sets
2:  Output: A GTSP tour on $G$.
3:  for $i = 1$ to $num\_trials$ do
4:      $T \leftarrow initial\_tour(G, P_V)$
5:      $T_{\text{best},i} \leftarrow T$
6:      repeat
7:          Select a removal heuristic $R$ and insertion heuristic $I$ using the selection weights
8:          Select the number of vertices to remove, $N_r$, uniformly randomly from $\{1, ..., N_{\max}\}$
9:          Create a copy of $T$ called $T_{\text{new}}$
10:         Remove $N_r$ vertices from $T_{\text{new}}$ using $R$
11:         For each of the $N_r$ sets not visited by $T_{new}$, using $I$ insert a new vertex into $T_{\text{new}}$
12:         Locally re-optimize $T_{\text{new}}$
13:         if $w(T_{\text{new}}) < w(T_{\text{best},i})$ then
14:             $T_{\text{best},i} \leftarrow T_{\text{new}}$
15:         end if
16:         if accept($T_{\text{new}}$, $T$) then
17:             $T \leftarrow T_{\text{new}}$
18:             Record improvement made by $R$ and $I$
19:         end if
20:     until stop criterion is met
21:     Update selection weights based on improvements of each heuristic over trial
22: end for
23: return tour $T_{\text{best},i}$ that attains $\min_i w(T_{\text{best},i})$
```

προβλήματος σε επί μέρους υποπροβλήματα και την επίλυση αυτών με το βέλτιστο δυνατό τρόπο.

Αρχικά, χρήσιμο είναι να παρουσιάσουμε τα δεδομένα που έχουμε στη διάθεσή μας για τη μετ΄ έπειτα ανάλυση που θα διεξάγουμε. Μετά από μία βόλτα του ρομπότ στον εξωτερικό χώρο του Κέντρου Υψηλής Τάσης, με τηλεχειρισμό αυτού, πραγματοποιήθηκε η λεγόμενη συλλογή δεδομένων. Από αυτήν προέκυψαν τα εξής. Χάρτες τοπολογικής αναπαράστασης του χώρου, δηλαδή με πληροφορίες του υψώματος σε κάθε σημείο, της τραχύτητας του εδάφους και άλλων αντίστοιχων παραμέτρων. Η Εικόνα 6 παρουσιάζει ένα χάρτη, με το χρώμα να υποδηλώνει την πληροφορία υψώματος. Να σημειωθεί ότι η πληροφορία αυτή αναφέρεται μόνο στη διαδρομή που εκτελέστηκε η κίνηση του ρομπότ στη προαναφερθείσα κίνηση συλλογής δεδομένων του. Υπάρχει και η αντίστοιχη αναπαράσταση με επέκταση για όλο το χάρτη, αλλά με κάποιο μεγαλύτερο σφάλμα.

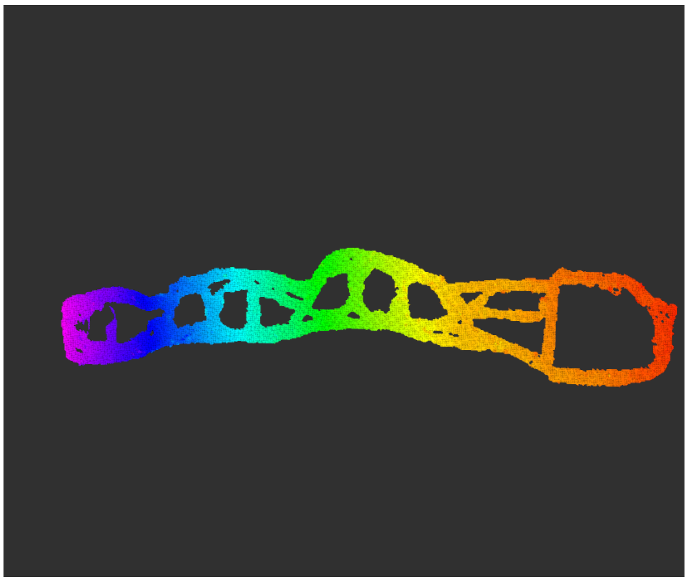


Figure 6. *Χάρτης αναπάραστασης του υψώματος σε κάθε σημείο της διαδρμομής που ακολούθησε το ρομπότ για τη συλλογή δεδομένων. Το χρώμα υποδεικνύει τις τιμές αυτές. Όσο κινούμαστε προς τα δεξιά πήγαινουμε σε υψηλότερες τιμές.*

Από τις παραπάνω πληροφορίες με κατάλληλη ανάλυση προκύπτουν χάρτες προσβασιμότητας/βατότητας, όπως της Εικόνας 7. Αυτή τη φορά παραθέτουμε την έκδοση ολόκληρου του χάρτη. Στην παρούσα φάση η πληροφορία αυτή είναι δυαδική, δηλαδή ή είναι προσβάσιμο ένα κελί ή όχι. Παρ΄ όλα αυτά η λύση μας λαμβάνει υπόψη τη μελλοντική επέκταση για συνεχές φάσμα τιμών προσβασιμότητας βάσει της ευκολίας ή δυσκολίας αντίστοιχα της πρόσβασης.

Επιπλέον, έχουμε στη διάθεση μας χάρτες που αναπαριστούν κόστη ορατότητας. Πόσο καλά δηλαδή βλέπει το κάθε κελί του δισδιάστου καρτεσιανού χώρου, τον κάθε πιθανό στόχο προς επιθεώρηση. Παράδειγμα ενδεικτικό είναι αυτό της Εικόνας 8. Με κόκκινο φαίνονται οι καλές περιοχές ενώ όσο πλησιάζει το κίτρινο, η πληροφορία που λαμβάνουμε από τα αντίστοιχα κελιά χειροτερεύει.

Αναπτύξαμε λοιπόν και υλοποιήσαμε μια **μεθοδολογία** για τη λύση του προβλήματος

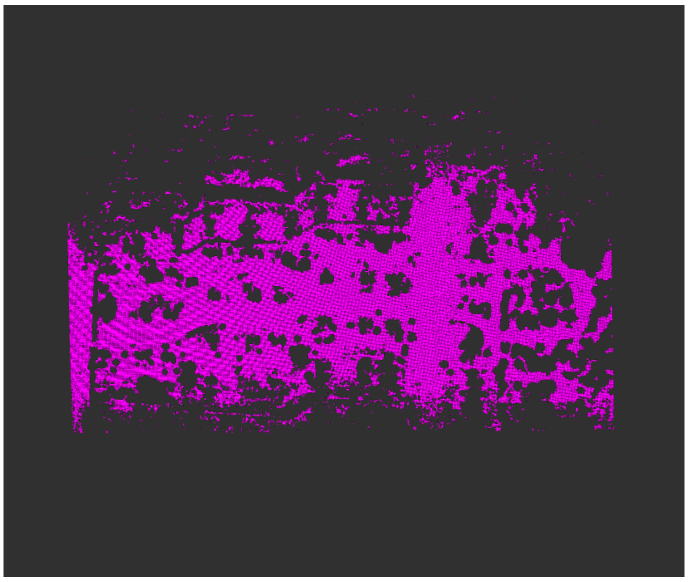


Figure 7. *Χάρτης αναπάραστασης της προσβασιμότητας όλου του εξωτερικού χώρου, γνωστός και με τον όρο traversability map.*

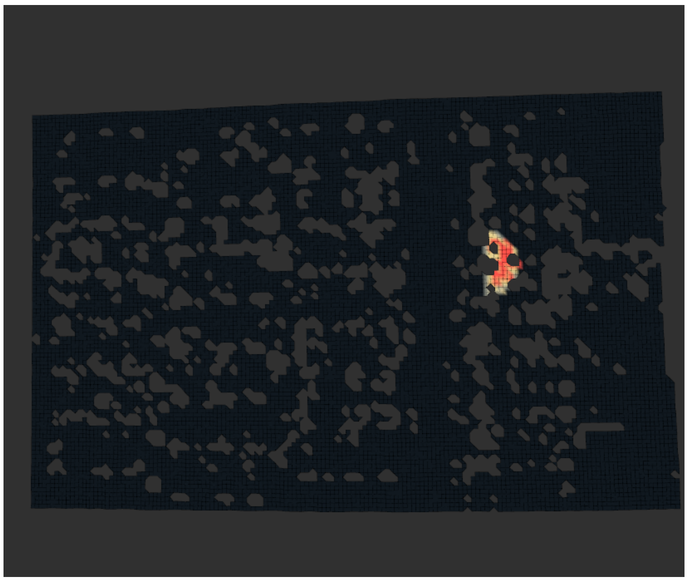


Figure 8. *Χάρτης αναπάραστασης της οπτικής πληροφορίας, για ένα στοιχείο στόχο. Στα αγγλικά αναφερόμαστε στο χάρτη ως visibility map.*

βάσει των δεδομένων που παρουσιάστηκαν παραπάνω. Η λύση μας βασίζεται σε δύο διαδικασίες. Η μία πραγματοποιείται μία φορά και προπαρασκευάζει πληροφορίες για αργότερα. Η δεύτερη πραγματοποιείται κάθε φορά ανάλογα με τις απαιτήσεις του χρήστη. Λεπτομέρειες μπορεί κανείς να δει στο διάγραμμα της Εικόνας 3.8, του αντίστοιχου κεφαλαίου.

Η πρώτη φάση αναφέρεται σε δύο βασικά στάδια. 1. Τον υπολογισμό όλων των πιθανών σημείων ενδιαφέροντος για κάθε στόχο επιθεώρησης (όλες τις καλές εικονοληπτικές θέσεις δηλαδή). 2. Τον υπολογισμό αποστάσεων μεταξύ όλων αυτών.

Για *τον υπολογισμό όλων των πιθανών σημείων ενδιαφέροντος για κάθε στόχο επιθεώρησης*, εργαζόμαστε ως εξής. Κάθε στοιχείο αναλύεται ξεχωριστά βάσει της οπτικής πληροφορίας που έχουμε στη διάθεσή μας. Ένα γκρουπ από σημεία/κελιά εξάγεται σαν αποτελέσμα για τον κάθε στόχο. Για να προκύψει αυτό αρχικά θέτουμε σε κάθε κελί τιμή ίση με το μέσο όρο των τιμών σε μία ακτίνα μικρή αλλά ικανοποιητική τριγύρω του. Αυτό πραγματοποείται προκειμένου τα σημεία ενδιαφέροντος να αναφέρονται σε μία καλή περιοχή και όχι σε μεμονομένα σημεία, που ενδεχομένως να υπάρχουν μετέπειτα δυσκολίες στις λήψεις. Ακολούθως, διατρέχουμε όλα τα κελιά και χρησιμοποιώντας τον αλγόριθμο introsort της standard libraryβιβλιοθήκης της γλώσσας C, τα σορτάρουμε. Σχεδιάσαμε και υλοποιήσαμε έναν άπληστο αλγόριθμο για την επιλογή των σημείων αυτών, ο οποίος δουλεύει ως εξής. Αρχικά επιλέγουμε το κελί με το καλύτερο σκορ/ένδειξη ορατότητας (μετά τη διαδικασία με τους μέσους όρους περιοχών). Ακολούθως, επιλέγουμε το αμέσως επόμενο καλύτερο το οποίο έχει μία ικανοποιητική απόσταση (πχ. 1 μέτρο) ακτίνα μακριά από όλα τα προηγούμενα. Η διαδικασία επαναλαμβάνεται έως ότου δεν υπάρχουν άλλα σημεία καλύτερα από ένα threshold και μακρύτερα από τα προηγούμενα βάσει των παραπάνω προδιαγραφών.

Στην Εικόνα 9 εξηγείται με τη βοήθεια σκαριφήματος, η αναγκαιότητα και το πως θα είναι οι λεγόμενες συστάδες σε ένα υποθετικό παράδειγμα. Στην Εικόνα 10, άλλωστε, παρουσιάζεται ένα πραγματικό παράδειγμα συστάδας σημείων για ένα σημείο-στόχο.

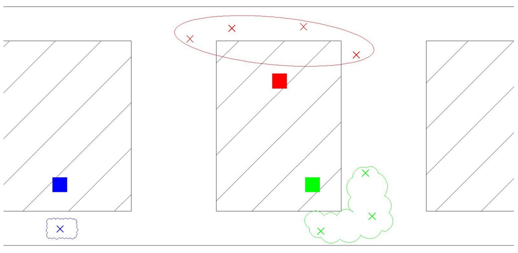


Figure 9. *Επεξηγηματική εικόνα για την αναγκαιότητα των γκρουπς/συστάδων. Με X απεικονίζονται τα καλά σημεία ορατότητας για τον κάθε στόχο.*

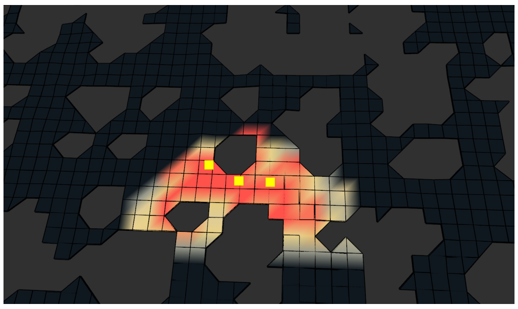


Figure 10. *Ένα αποτέλεσμα συστάδας. Φαίνεται από τα κίτρινα κουτάκια, πάνω στο χάρτη ορατότητας που επεξηγήθηκε προηγουμένως.*

Στο σημείο αυτό, ενώ έχουν εξαχθεί όλα τα πιθανά σημεία ενδιαφέροντος, επιθυμούμε να *υπολογίσουμε όλες τις μεταξύ τους αποστάσεις* (offline). Η κύρια απαίτηση στην παρούσα φάση είναι ο καλύτερος δυνατός υπολογισμός όλων των αποστάσεων αυτών, με το μικρότερο δυνατό σφάλμα δηλαδή, καθώς οι πληροφορίες που θα προκύψουν θα χρησιμοποιούνται συνέχεια στο μέλλον. Για να επιτευχθεί αυτό, κρίθηκε ορθή η επιλογή του αλγορίθμου A*, με ευριστική συνάρτηση την ευκλίδεια απόσταση (που οδηγεί σε βέλτιστες λύσεις, καθώς είναι πάντα μικρότερη ή ίση με την πραγματική απόσταση ως το στόχο), και η επαναληπτική εφαρμογή αυτού μέχρι εξαντλήσεως όλων των πιθανών ζευγών από σημεία. Αποτελέσμα της όλης διαδικασίας είναι η εξαγωγή ενός πίνακα με δεδομένα αποστάσεων.

Όταν ο χρήστης του προγράμματός μας επιλέξει τα στοιχεία που θέλει να επιθεωρηθούν αυτόματα από το κινούμενο ρομπότ, μια σειρά από δράσεις λαμβάνουν χώρα. Βάσει των σημείων ενδιαφέροντος και του πίνακα αποστάσεων που έχουν παραχθεί από τα προηγούμενα βήματα, δημιουργούμε το κατάλληλο αρχείο για την επίλυση του Γενικευμένου Προβλήματος Πλανόδιου Πωλητή GTSP. Το εν λόγω αρχείο, έχει φορμάτ όπως φαίνεται παρακάτω.

Listing 1: GTSP File Format for Solvers

```
NAME: GTSP_format_explanation
TYPE: GTSP
DIMENSION: 6
GTSP_SETS: 3
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 200 2310 6052 404 10912
240 0 2166 5895 634 10719
2326 2195 0 3900 2724 8801
6195 6031 3978 0 6599 4888
434 639 2723 6457 0 11356
10869 10598 8697 4725 11217 0
GTSP_SET_SECTION
1 1 3 -1
2 2 4 5 -1
3 6 -1
EOF
```

Όπως φαίνεται παραπάνω δηλώνονται τα κόστη μετάβασης από κάθε σημείο ενδιαφέροντος σε κάθε άλλο. Κάτω ορίζονται οι συστάδες. Στο παρόν παράδειγμα, έχουμε 3 συστάδες, η 1η αποτελείται από τα σημεία υπ' αριθμόν 1 και 3, η 2η από {2,4,5}, ενώ η 3η μόνο από το 6ο κατά σειρά σημείο.

Αυτό το φορμάτ διαβάζεται από τα διαθέσιμα προγράμματα επίλυσης GTSP προβλημάτων, και επιστρέφουν την αλληλουχία από σημεία για βέλτιστη δυνατή λύση του γενικευμένου προβλήματος πλανόδιου πωλητή.

Στο σημείο αυτό δοκιμάσαμε τους 2 καλύτερους solver, GLKH & GLNS, όπως παρουσιάστηκαν από τη σχετική βιβλιογραφική έρευνα. Λόγω της γενικότερης υπεροχής του GLNS, στη συγκριτική αξιολόγησή τους, όπως παρουσιάζεται και στην Εικόνα 5, αυτή ήταν και η υλοποίηση που τελικώς επιλέχθηκε. Αξίζει να σημειωθεί πως η προαναφερθείσα σύγκριση έχει εκτελεστεί σε πολύ μεγάλα datasets προβλημάτων ειδικά σχεδιασμένα γι' αυτό το σκοπό. Τα μικρά σε όγκο δεδομένα που έχουμε στα πλαίσια αυτής της διπλωματικής παρουσιάζουν παρόμοια επίδοση και για τους δύο λύτες, καθώς πάντα βρίσκουν τη βέλτιστη αλληλουχία και σε πολύ παρόμοιο χρόνο. Γι΄ αυτό κρίνεται άσκοπη η εξαγωγή συμπερασμάτων κατ΄ αυτόν τον τρόπο.

Πραγματοποιήθηκε ποιοτικός έλεγχος των αποτελεσμάτων μας, ο οποίος βοήθησε πολύ κατά την ανάπτυξη του κώδικα σε διόρθωση λαθών και στην τελική επιβεβαίωση ότι η όλη η διαδικασία δουλεύει όπως περιμέναμε. Η Εικόνα 11, παρουσιάζει ένα παράδειγμα βέλτιστης προτεινόμενης αλληλουχίας αλλά και της διαδρομής που προτείνει ο global planner. Έχει πραγματοποιηθεί inflation στο χάρτη γύρω από εμπόδια και γενικότερα μη προσβάσιμα σημεία, ώστε το ρομπότ να οδηγείται μέσα από πιο ασφαλείς γειτονιές σημείων. Με κίτρινο παρουσιάζονται τα σημεία ενδιαφέροντος, ενώ με την πράσινη γραμμή η διαδρομή που προτείνεται. Ο local planner στη συνέχεια βασιζόμενος σε αυτό το σκέπτικο θα αξιοποιήσει τους κατάλληλους αλγορίθμους, ώστε μόνο με είσοδο τα σημεία (συντεταγμένες) και την κατάλληλη αλληλουχία αυτών, και όχι την πράσινη γραμμή, να εκτελέσει τη συνολική κίνηση για την εργασία επιθεώρησης.

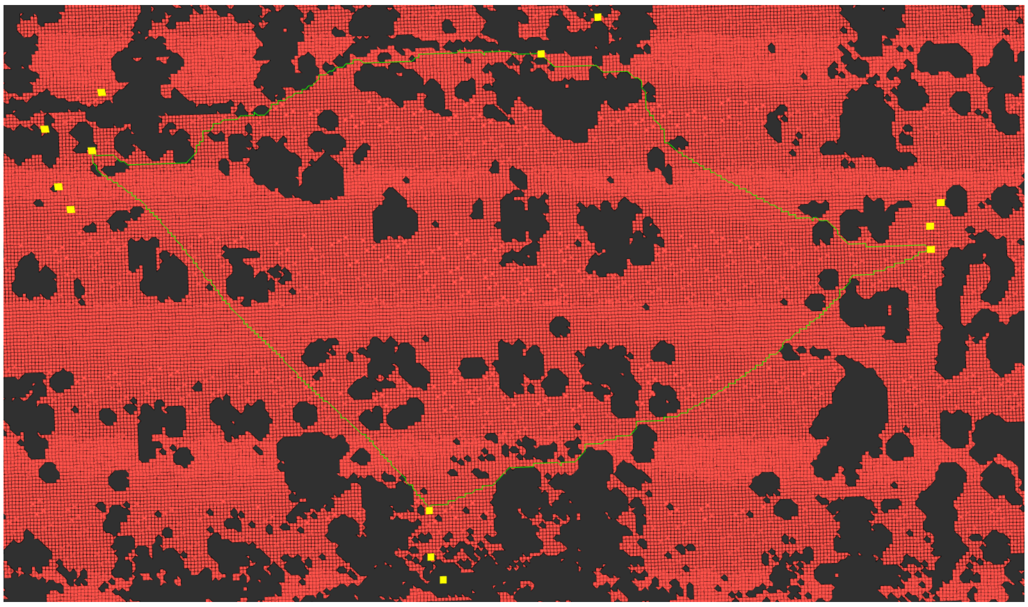


Figure 11. *Παράδειγμα αποτελέσματος μιας βέλτιστης διαδρομής για την επιθεώρηση 4 στοιχείων.*

Για την *υλοποίηση* των παραπάνω χρησιμοποιήθηκαν οι εξής τεχνολογίες. Αρχικά, γλώσσα προγραμματισμού C++ ώστε να έχουμε την καλύτερη δυνατή απόδοση. Αξιοποιήθηκε το ROS framework και βοηθητική φάνηκε η χρήση μερικών συναρτήσεων από τις βιβλιοθήκες gridmap, costmap, navfn, κυρίως στη διαχείριση των δεοδμένων του χάρτη. Οι υλοποιήσεις των GTSP Solvers, GLKH & GLNS, είναι διαθέσιμες ευρέως στο διαδίκτυο και ενσωματώθηκαν με τον κατάλληλο τρόπο. Τέλος, αξιοποιήθηκε το εργαλείο RViz ROS για την οπτικοποίηση όλων των αποτελέσματων, που είδαμε και στην παραπάνω ανάλυση. Πραγματοποιήθηκε εκτενής ποιοτική ανάλυση των αποτελεσμάτων, η οποία παρουσιάζεται στο αντίστοιχο κεφάλαιο, επιβεβαιώνοντας την ορθή λειτουργία της λύσης που προτείνουμε.

4. Συμπεράσματα και μελλοντική έρευνα

Συμπερασματικά, υλοποιήθηκε μία μέθοδος για τη λύση ένος προβλήματος βελτιστοποίησης πολλαπλών κριτηρίων για την επιθέωρηση στοιχείων με εφαρμογή σε Κέντρα Υψηλής Τάσης. Κομμάτια από την προτεινόμενη λύση μπορούν να επεκταθούν και να εφαρμοστούν σε πληθώρα άλλων εφαρμογών και προβλημάτων, από σενάρια αναζήτησης και διάσωσης, σε ρομποτικές κατασκευές και συγκολλήσεις, μέχρι σχεδίαση ολοκληρωμένων κυκλωμάτων, αλλά ακόμα και εφαρμογές σε τουρισμό και επίσκεψεις αξιοθέατων.

Εννοείται πως το έργο μας επιδέχεται επεκτάσεις και παρουσιάζει μεγάλο ενδιαφέρον η κατεύθυνση αυτή. Ενδεικτικά προτείνουμε την έρευνα και ανάπτυξη ενός καλύτερου αλγορίθμου, από τον άπληστο αλγόριθμο που υλοποιήθηκε, για την εξαγωγή σημείων ενδιαφέροντος. Επιθυμούμε τη μεγιστοποίηση του αριθμού των πιθανών σημείων, που παράλληλα έχει το βέλτιστο συνολικό μέσο όρο από σκορς ορατότητας. Μεγάλο ερευνητικό ενδιαφέρον άλλωστε παρουσιάζει, η ανάπτυξη αντίστοιχης λύσης για συνεργαζόμενο γκρουπ από ρομπότ. Η ανάπτυξη μίας τέτοιας λύσης θα βοηθήσει πολύ σε επιθεωρήσεις μεγαλύτερες κλίμακας και σε εκτενέστερα και πιο περίπλοκα περιβάλλοντα.

Chapter 1

Introduction

1.1 Basic Definitions

Mobile robotics is a subfield of robotics that is concerned with the design, construction and operation of robots that are not fixed in a certain physical location, but have the capability to move around in various environments. Mobile robots, unlike industrial ones that typically operate within confined spaces, can navigate through dynamic and often unpredictable surroundings. These robots are usually equipped with onboard sensors, actuators and proper computer programs/intelligence in order to perceive their environment, make decisions and adapt autonomously to changing conditions.

Inspection, within the context of robotics, refers to the examination, assessment and generally surveillance of objects, structures or even whole environments using robotic systems. Robotic inspection serves different purposes across industries including infrastructure maintenance, environmental monitoring and security surveillance. Robots equipped with proper sensors can navigate autonomously and perform the inspection efficiently and accurately in a repeated fashion or in hazardous/hard-to-reach locations.

Path planning, also known as motion planning, is a computational problem of finding a sequence of valid configurations from which a robot can navigate from a current position to a specified goal within an environment. The objective, most of the times, is for the robot to reach its destination safely and efficiently; minimizing the risk of possible collisions (or minimizing/maximizing other metrics), while at the same time optimizing its route to conserve energy and time.

An optimization problem, in mathematics, engineering, computer science and economics, is the problem of finding the best solution from all feasible solutions. The Traveling Salesman Problem (TSP) is one classic optimization problem which asks the question: "Given a list of cities and the distances between them, what is the shortest possible route that visits each city exactly once and returns to the original city?" This problem extends to different scenarios and will be later more thoroughly discussed in our study.

1.2 Problem

The problem we tackle is as follows. There is a set of different items we want to inspect, by taking photos (both RGB and infrared), in an outdoor environment. We possess a topological representation of the static map of this environment, along with a map indicating how well various poses across the map perceive each one of these items. We are equipped with a 4-wheel mobile robot, using LiDAR, different other sensors like IMU (Inertial Measurement Unit) and the necessary actuators for autonomous navigation. A user determines which elements from the environment require inspection on each occasion.

Consequently, an automated solution is needed to generate the most optimal plan and sequence for visiting suitable configurations capable of effectively inspecting every user-selected element (taking into consideration both the time and energy limits, while simultaneously capturing the best possible images for inspection).

The solution of this problem will be applied within the context of a project called "ENO-RASI", which deals with inspecting elements of high-voltage facilities of the ADMIE/IPTO (Independent Power Transmission Operator).

Figure 1.1. *Outdoor environment of a high voltage facility in Pallini, Greece.*

Figure 1.2. *High voltage facility, some elements that may need inspection shown.*

1.3 Our Contribution

Taking into consideration the above, we propose an automated system that takes as inputs a map representation of the environment, a cost-map that indicates how well each pose captures the inspection and a set of elements requiring inspection. After that the program suggests the best sequence of points that it needs to go so as to carry out the inspection task. This sequence will then be executed onboard using a local planner.

Our proposal is a system that creates an offline distance matrix between different points that may be useful in the future. When the user selects the elements to be inspected, a set of points is generated for each one. These points, located in cartesian space, indicate that the visibility of the element/item there exceeds a certain threshold. The system employs an EGTSP (Equality Generalized Traveling Salesman Problem) solver, called GLNS, to suggest the best possible sequence of points in the environment to visit that optimizes the inspection task.

1.4 Organization

The present thesis is structed as follows:

- In Chapter 2 we analyze related work, background knowledge and tools.
  - Chapter 2.1 deals with key terminology,
  - Chapter 2.2 presents the path planning problem and describes some solutions.
  - Chapter 2.3 focuses on the Traveling Salesman Problem, its approaches and presents some solutions of it.
  - Chapter 2.4 extends the analysis to the Generalized Traveling Salesman Problem.
  - Chapter 2.5 briefly presents available technologies.
- In Chapter 3 we thoroughly present our contribution and solution to the problem.
- In Chapter 4 we present the experimental results of our solution.
- In Chapter 5 we summarize the key concepts of the work and discuss future research directions.

Chapter 2

Background & Related Work

2.1 Key Terminology

In this section, we delve into key terminology relevant to our study, providing a comprehensive understanding of the fundamental concepts needed.

Optimization refers to the process of finding the best solution among a set of feasible solutions to a particular problem. This could involve maximizing or minimizing an objective function while satisfying certain constraints. *Multi-objective optimization*, as the name suggests, involves optimizing several objectives simultaneously. The problem becomes challenging when the objectives are of conflict to each other, that is, the optimal solution of an objective function is different from that of the other. With or without the presence of constraints, these problems give rise to a set of trade-off optimal solutions, popularly known as *Pareto-optimal solutions*.

Combinatorial optimization is a subfield of mathematical optimization focused on finding an optimal object from a finite set of objects, where the set of feasible solutions is discrete or can be reduced to a discrete set.

Complexity in the context of computational problems refers to the amount of computational resources required to solve a problem. It is commonly categorized as time complexity, which measures the number of computational steps needed to solve a problem, and space complexity, which measures the amount of memory required for the solution.

The *Big O* notation, often denoted as $O()$, is a mathematical notation used in computer science to exactly describe the upper bound complexity of an algorithm. It represents the upper bound of the growth rate of a function, typically in terms of the input size n, where n represents the size of the problem.

A *polynomial algorithm* is one whose time complexity is bounded by a polynomial function of the input size. Problems with polynomial time complexity are generally considered efficiently solvable.

NP problems are a class of computational problems for which solutions can be verified in polynomial time. This means that if a solution is given, it can be relatively quickly checked. However, finding a solution may be much more difficult. As [20] thoroughly explains, *NP-hard* problems are those for which a solution can be verified in polynomial time but for which no polynomial-time algorithm is known for finding a solution. *NP-complete* problems are a subset of NP-hard problems. They are the most challenging among NP problems because they are both in NP and NP-hard, meaning that any problem in NP can be reduced to an NP-complete problem in polynomial time. In essence, if an efficient algorithm for solving any NP-complete problem exists, it could be used to solve all NP problems efficiently.

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph is made up of vertices, also called nodes, which are connected by edges, also called links. A distinction is made between undirected graphs, where edges link two vertices symmetrically, and directed graphs, where edges link two vertices asymmetrically. Graphs are one of the principal objects of study in *discrete mathematics*.

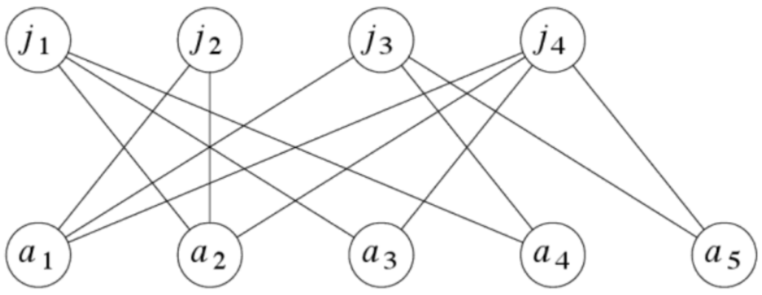


Figure 2.1. *A job applicant model, depicted as a graph [3]*

.

A *tree* is a connected graph without cycles. A *spanning tree* of a graph G with n nodes is a tree with n-1 edges from G. A *minimum spanning* tree is a spanning tree of minimum length.

A *Hamiltonian path*, in graph theory, is a path in an undirected or directed graph that visits each vertex exactly once. A *Hamiltonian cycle* is a cycle that visits each vertex exactly once.

Heuristic functions, fundamental in artificial intelligence and optimization, are problem-solving tools which estimate (or measure) the cost of the solution at the particular state in the search process. These functions guide by offering a way to evaluate possible actions or paths in a problem space, assisting in decision-making. Heuristics are particularly valuable in scenarios where exhaustive search methods are impractical due to high computational complexity. By leveraging heuristic information, algorithms can efficiently navigate large search spaces to find satisfactory solutions in a timely manner. A *heuristic algorithm* is an algorithm that attempts to find a certain instance of X that maximizes f (or the profit) by iteratively invoking a heuristic function. The instance that maximizes f will be the optimal solution to the optimization problem. [21]

Many heuristic algorithms and heuristic functions have been reported in the literature, where the former include the alpha-beta search [22], backtracking, hill-climbing [23], simulated annealing [24], tabu search [25] and other.

Greedy algorithms are simple, yet powerful, techniques used to solve optimization problems by making the locally optimal choice at each stage. They select the best available option without considering the overall problem structure or future consequences.

Visibility refers to the extent to which an object or location can be seen from a particular point or set of points. In the context of robotics and path planning, typically visibility plays a crucial role in determining the feasibility and effectiveness of navigation strategies. In our context, visibility will play a vital role in determining the effectiveness of our path planning algorithms for the inspection task we are dealing with.

Mobile ground robots are traditionally designed to move on flat terrain and their mapping, planning, and control algorithms are typically developed for a two-dimensional abstraction of the environment. However, when navigating in rough terrain (e.g. with tracked vehicles or legged robots), the algorithms must be extended to take into account all three dimensions of the surrounding. As [4] thoroughly describes, the most popular approach is to build an *elevation map* of the environment, where each coordinate on the horizontal plane is associated with an elevation/height value. For simplicity, elevation maps are often stored and handled as grid maps, which can be thought of as a 2.5-dimensional representation, where each cell in the grid holds a height value. In order to estimate the *traversability* of the terrain, the elevation information is appropriately processed; multiple filters are used to interpret the data based on the slope, step height, and roughness of the terrain.

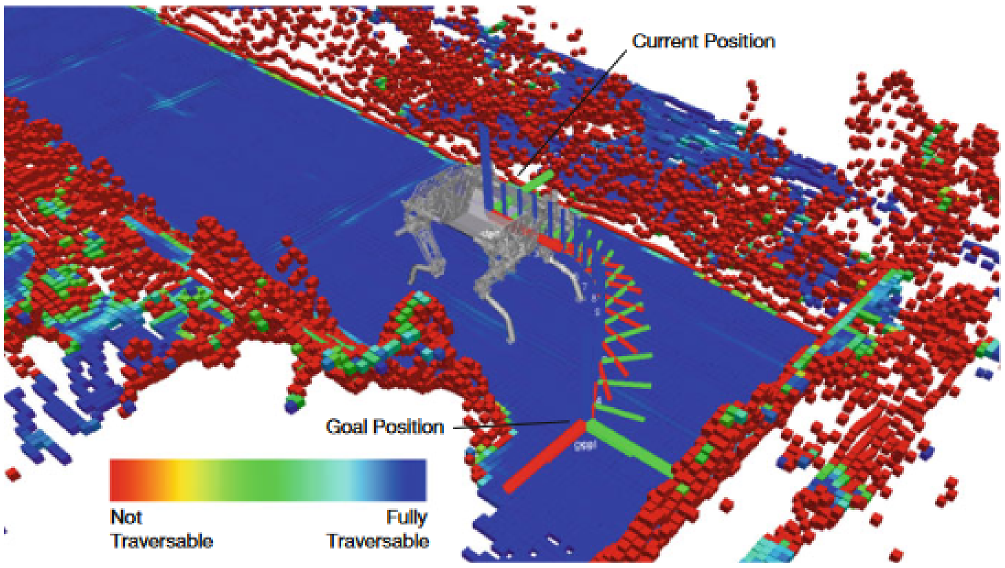


Figure 2.2. *The traversability of the terrain is judged based on the acquired elevation map. The traversability estimation takes factors such as slope, step size, and roughness of the terrain into consideration. [4]*

A *LiDAR sensor* (acronym stands for "light detection and ranging") commonly uses a mechanically rotating laser beam to sweep a planar sector of the environment, measuring time-of-flight of the reflected beams to compute range estimates from the sensor to obstacles. *IMU* (Inertial Measurement Unit) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. *Wheel encoders* measure the rotation of the vehicle's wheels, providing some information about its movement, speed, and distance traveled. These are the main sensors that combined with cameras are frequently used to calculate the map representation of the environment, help the robot localize and navigate autonomously.

2.2 Path Planning Algorithms

Path planning is a fundamental task in robotics, essential for letting autonomous vehicles and generally robots navigate from an initial state to a desired destination through an environment. The definition of the path planning problem is very straightforward: “find a collision-free motion between an initial (start) and a final configuration (goal) within a specified environment” [11]. Simultaneously the extracted solution can also optimize certain criteria such as minimizing the total distance covered, maximizing safety or other task-related criteria (e.g. maximizing visibility metrics in an inspection task).

Much work can be found the robotic literature, dealing with path planning. The first definitions and algorithms date back to the 1970s. A comprehensive overview of path planning techniques can be found in [7]. Other useful reviews of path planning techniques can be found in [8, 9, 10].

Path planning algorithms are usually divided in three categories, according to the methodologies used to generate the geometric path, namely: roadmap techniques, cell decomposition algorithms, and artificial potential methods [11].

The difficulty in solving the path-planning problem rises from factors like high dimensional search spaces, geometric constraints or limitations in computational resources. Therefore the solution often requires a trade-off between efficiency and optimality. Path planning methods can be broadly categorized into offline and online approaches [26], each one handling differently this trade-off. Offline planners compute the entire trajectory beforehand, whereas online planners generate it incrementally during motion. Both approaches have their own advantages and are suited to different applications. Offline planning is useful for tasks in static environments where optimality is crucial, while online planning is essential for dynamic environments or scenarios where the robot’s path must be determined on-the-go. Another distinction between offline and online planners is that the former may produce globally optimal solutions if the environment is fully known, whereas the latter is locally optimal most of the times.

For the local planning problems one of the most used algorithms proposed is the RRT (Rapidly exploring random trees). For the global planning problem, that we also face in our work, the two most prominent methods are the Dijkstra’s algorithm and the A* algorithm. In the following sections we present them in detail.

2.2.1 Dijkstra’s Algorithm

Dijkstra’s algorithm, named after its discoverer E.W. Dijkstra [27], is a graph search algorithm that solves the shortest-path problem for any weighted graph with non-negative weights [12]. It starts from a source node and iteratively explores the neighboring nodes, updating the shortest distance to each node until all nodes have been visited. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

Dijkstra's original algorithm does not use a min-priority queue and runs in time $\Theta(|V|^2)$ [28]. The time complexity of Dijkstra's algorithm, implemented with a binary heap, is O(|E|+|V|log|V|), *where* V are the vertices and E are the edges [29]. There are various other implementations of Dijkstra's algorithm, each one having slightly different time complexity. The selection obviously is based on the specific needs of each individual problem.

2.2.2 A* Algorithm

A* (pronounced as "A star") is a computer algorithm that is widely used in path-finding and graph traversal. Given a weighted graph, a source node and a goal node, the algorithm tries to find the shortest path (with respect to the given weights) from source to goal. A* considers both the actual cost from the start node (source) to the current node and the estimated cost (heuristic) to reach the goal from this current node, where current node refers to an intermediate node in the path. Thus, it intelligently prioritizes nodes to explore, leading to an optimal path with minimal computational overhead. Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first published the algorithm in 1968 [13].

At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes: $f(n) = g(n)+h(n)$, where n is the next node on the path, *g(n)* is the cost of the path from the start node to n, and *h(n)* is a heuristic function that estimates the cost of the cheapest path from n to the goal.

Compared to Dijkstra's algorithm, A* typically explores fewer nodes than Dijkstra's algorithm, especially in scenarios where the heuristic provides meaningful information about the distance to the goal. Yet, the A* algorithm is limited to finding the shortest path from a given source to a particular goal and cannot construct the shortest-path tree from the source to all potential destinations. This constraint is necessary for using a specific-goal-directed heuristic.

As [13] throughly discusses, we call an algorithm admissible if it is guaranteed to find an optimal path from source to a preferred goal node for any graph. It is proved that if the heuristic function selected for A* suggests costs that are always smaller or equal to the actual distance from the current node to the goal node, that the algorithm always will suggest the optimal, best path, and thus be considered admissible.

As [30] describes in detail, the time complexity of A* depends on the heuristic. In the worst case of an unbounded search space, the number of expanded nodes grows exponentially with the depth of the solution (the shortest path), denoted by d: O(b^d), where b is the branching factor (the average number of successors per state). This analysis assumes the presence of a goal state and its accessibility from the initial state. However, if the goal state is unreachable from the start state and the state space is infinite, the algorithm will not terminate. The heuristic function has a vital effect on the practical performance of A* search, since a good heuristic allows A* to prune away many of the b^d nodes that an uninformed search would expand. The time complexity is polynomial when the search space is a tree, and the heuristic function h meets the following condition:

$$|h(x) - h*(x)| = O(logh*(x))$$

where h* is the exact cost to get from x to the goal, also referred to as the optimal heuristic. In simpler terms, the error of the heuristic function h will not grow faster than the logarithm of the h* that returns the true distance from x to the goal.

The space complexity of A* is approximately the same as that of all other graph search algorithms, as it keeps all generated nodes in memory. In practice, this aspect turns out to be the biggest disadvantage of the A* search algorithm, prompting the development of memory-bounded heuristic search methods like Iterative Deepening A* [31].

2.3 Traveling Salesman Problem

The Traveling Salesman Problem, TSP for short, is a well-known optimization problem that has commanded much attention of mathematicians and computer scientists due to its simple yet challenging nature. The problem can simply be stated as: If a traveling salesman wishes to visit exactly once each of a list of m cities (where the cost of traveling from city i to city j is c_{ij}) and then return to the home city, what is the most efficient route the salesman can take to minimize overall travel cost, returning to the starting city?

The Traveling Salesman Problem is quite famous with various applications in different scenarios. The problem emerges in manufacturing of computer chips, in the order-picking problem in warehouses, in mask plotting in PCB (Printed Circuit Board) production, to even x-ray crystallography, as outlined in [14]. As [2] states, the problem has been recorded in the history of combinatorial optimization since 1930. Merrill M. The first mathematical formulation of the problem was provided by Flood when he came across it while solving a school bus routing problem in 1948 [32].

The challenge becomes evident when one considers the number of potential tours - an staggering figure even for a relatively small number of "cities". For a symmetric problem with n cities it can easily be proven that there are (n-1)!/2 possible tours. So, if n is 20, there are more than 10^{18} tours.

As the theory of NP-completeness evolved, the TSP was one of the first problems to be proven NP-hard by Karp in 1972. Consequently, new algorithmic techniques have first been developed for or at least have been applied to the TSP to showcase their effectiveness. Some examples are branch and bound, Lagrangean relaxation, Lin-Kernighan type methods, simulated annealing, and the field of polyhedral combinatorics for hard combinatorial optimization problems [33].

Algorithms for solving the TSP may be divided into two main classes:

- Exact algorithms;
- Approximate (or heuristic) algorithms.

Exact algorithms are assured to discover the optimal solution within a finite number of steps. Today one can find exact solutions to symmetric problems with a few hundred cities, with occasional reports even mentioning solutions involving problems with thousands of cities. The most effective exact algorithms are cutting-plane or facet-finding algorithms [34, 35]. The complexity of these algorithms is notable, often consisting of codes on the order of 10,000 lines. Additionally, the algorithms are very demanding of computer power. For instance, achieving an exact solution for a symmetric problem involving 2392 cities took over 27 hours on a high-performance supercomputer [36].

In contrast, the approximate algorithms yield satisfactory solutions without ensuring the discovery of the optimal ones. These algorithms are usually very simple and have (relative) short running times. Some of the algorithms give solutions that differ only by a few percents from the optimal ones, in average. Therefore, if a small deviation from optimum is acceptable, it might be appropriate to use an approximate algorithm.

The class of approximate algorithms may be subdivided into the following three classes [5]:

- Tour construction algorithms
- Tour improvement algorithms
- Composite algorithms

The tour construction algorithms operate by incrementally constructing a tour, adding one city at a time until the tour is complete. The tour improvement algorithms improve upon a tour by performing various exchanges.

In the next subsections, we will shortly delve into some of the most prominent algorithmic approaches to solving the problem. This analysis will also help build a sufficient knowledge-base for the understanding of later algorithms that are discussed in this work.

2.3.1 Nearest-neighbor Algorithm

A simple example of a tour construction algorithm is the so-called nearest-neighbor algorithm [37]: Start in an arbitrary city. While there are still unvisited cities, visit the nearest city that still has not appeared in the tour. Finally, return to the first city. This approach is simple, but often too greedy. The first distances in the construction process are reasonable short, whereas the distances at the end of the process usually will be rather long. A lot of other construction algorithms have been developed to remedy this problem (see for example [38] and [39]).

2.3.2 2-opt Algorithm

The tour improvement algorithms have shown significant success, with one notable example being the 2-opt algorithm. This approach begins with a predefined tour and iteratively replaces two links within the tour with two different links, resulting in a shorter overall tour length. The process continues until no further improvements can be made.

Figure 2.3 illustrates a 2-opt exchange of links, a so-called 2-opt move. Note that a 2-opt move keeps the tour feasible and corresponds to a reversal of a subsequence of the cities.

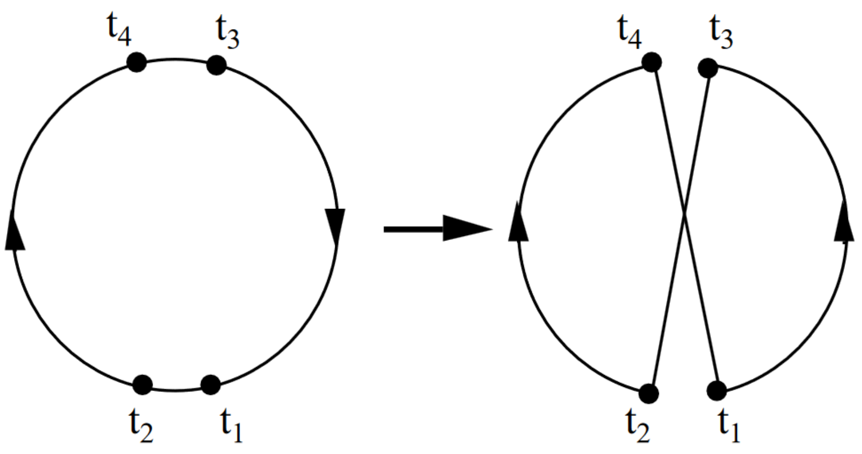


Figure 2.3. *A 2-opt move [5]*

2.3.3 Lin-Kernighan Algorithm

A generalization of this simple principle forms the basis for one of the most effective approximate algorithms for solving the symmetric TSP, the Lin-Kernighan algorithm [6]. The original algorithm, as implemented by Lin and Kernighan in 1971, had an average running time of order $n^{2.2}$ and was able to find the optimal solutions for most problems with fewer than 100 cities.

As [6] thoroughly describes, the 2-opt algorithm is a special case of the λ-*opt algorithm* [40], where in each step λ links of the current tour are replaced by λ links in such a way that a shorter tour is achieved. In other words, in each step a shorter tour is obtained by deleting λ links and putting the resulting paths together in a new way, possibly reversing one ore more of them.

A tour is said to be λ-optimal (or simply λ-opt) if it is impossible to obtain a shorter tour by replacing any λ of its links by any other set of λ links. It is obvious that any λ-optimal tour is also λ'-optimal for $1 \leq \lambda' \leq \lambda$. It is also easy to see that a tour that contains n cities is optimal if and only if it is n-optimal. It is a drawback that λ must be specified in advance. It is difficult to know what λ to use to achieve the best compromise between running time and quality of solution.

Lin and Kernighan addressed this limitation by introducing a powerful variable λ-opt algorithm. This algorithm dynamically adjusts the λ value as it progresses, determining the appropriate λ value at each iteration of its execution. At each iteration step the algorithm examines, for ascending values of λ, whether an interchange of λ links may result in a shorter tour. Given that the exchange of r links is being considered, a series of tests is performed to determine whether r+1 link exchanges should be considered. This process continues until certain termination criteria are met.

The Lin-Kernighan algorithm falls under the category of local optimization algorithms [41]. It operates by executing exchanges, also known as moves, that transform one tour into another. Given a feasible tour, the algorithm repeatedly performs exchanges to minimize the length of the current tour, until a tour is reached for which no exchange yields an improvement. This process can be repeated numerous times starting from initial tours generated in a randomized manner.

Below, we provide a detailed description of the algorithm [6]:

Let T be the current tour. At each iteration step the algorithm attempts to find two sets of links, $X = \{x_1, ..., x_r\}$ and $Y = \{y_1, ..., y_r\}$, such that, if the links of X are deleted from T and replaced by the links of Y, the result is a better tour. This interchange of links is called a r-opt move. Figure 2.4 illustrates a 3-opt move.

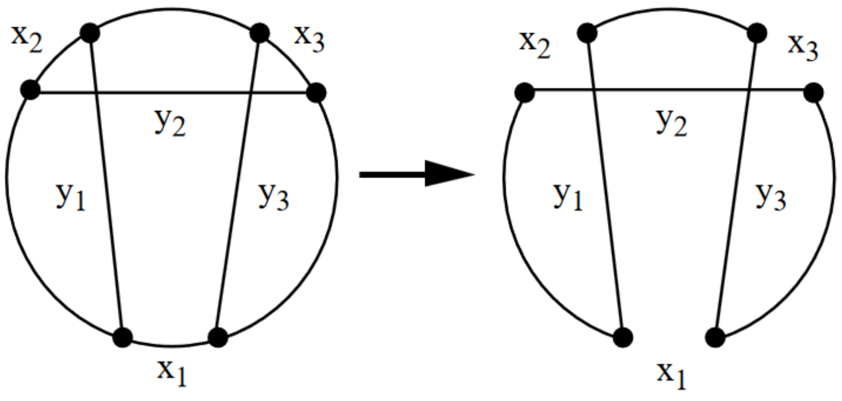


Figure 2.4. *A 3-opt move [5]*

Initially both X and Y sets are empty and they are being constructed element by element. In step i, a pair of links, x_i and y_i, are added to X and Y, respectively. In order to achieve a sufficient efficient algorithm, only links that fulfill the following criteria may enter X and Y.

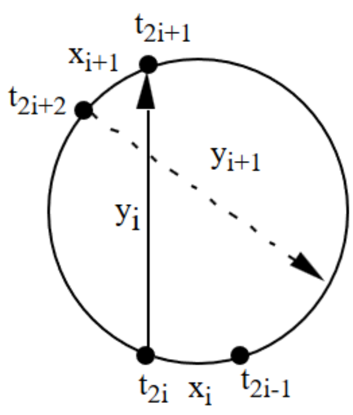


Figure 2.5. *Restricting the choice of x_i, y_i, x_{i+1}, and y_{i+1} [6]*

1. *The sequential exchange criterion:*

   Links x_i and y_i must share an endpoint, and so must y_i and x_{i+1}. If t_1 denotes one of the two endpoints of x_1, we have in general: $x_i = (t_{2i-1}, t_{2i})$, $y_i = (t_{2i}, t_{2i+1})$, and $x_{i+1} = (t_{2i+1}, t_{2i+2})$ for $i \geq 1$. See Figure 2.5.

   As seen, the sequence $(x_1, y_1, x_2, y_2, x_3, ..., x_r, y_r)$ constitutes a chain of adjoining links.

A necessary (but not sufficient) condition that the exchange of links X with links Y results in a tour is that the chain is closed, i.e., $y_r = (t_{2r}, t_1)$. Such an exchange is called sequential.

Generally, an improvement of a tour may be achieved as a sequential exchange by a suitable numbering of the affected links. However, this is not always the case. Figure 2.6 shows an example where a sequential exchange is not possible.

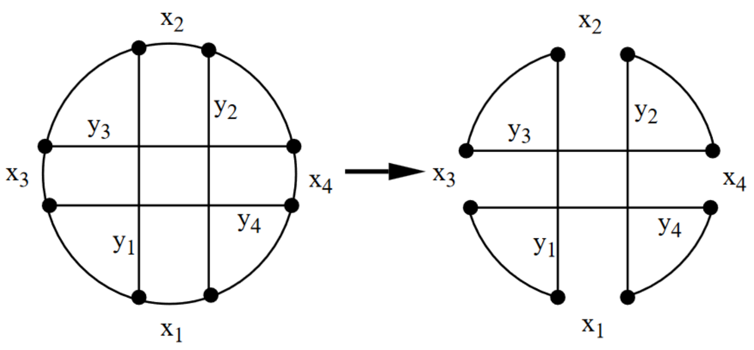


Figure 2.6. *Nonsequential exchange (r=4) [6]*

2. *The feasibility criterion:*

   It is required that $x_i = (t_{2i-1}, t_{2i})$ is chosen so that, if t_{2i} is joined to t_1, the resulting configuration is a tour. This feasibility criterion is used for $i \geq 3$ and guarantees that it is possible to close up to a tour. This criterion was included in the algorithm both to reduce running time and to simplify the coding.

3. *The positive gain criterion:*

   It is required that y_i is always chosen so that the gain, G_i, from the proposed set of exchanges is positive. Suppose $g_i = c(x_i) - c(y_i)$ is the gain from exchanging x_i with y_i. Then G_i is the sum $g_1 + g_2 + \ldots + g_i$. This stop criterion impacts the algorithm's overall efficiency.

4. *The disjunctivity criterion:*

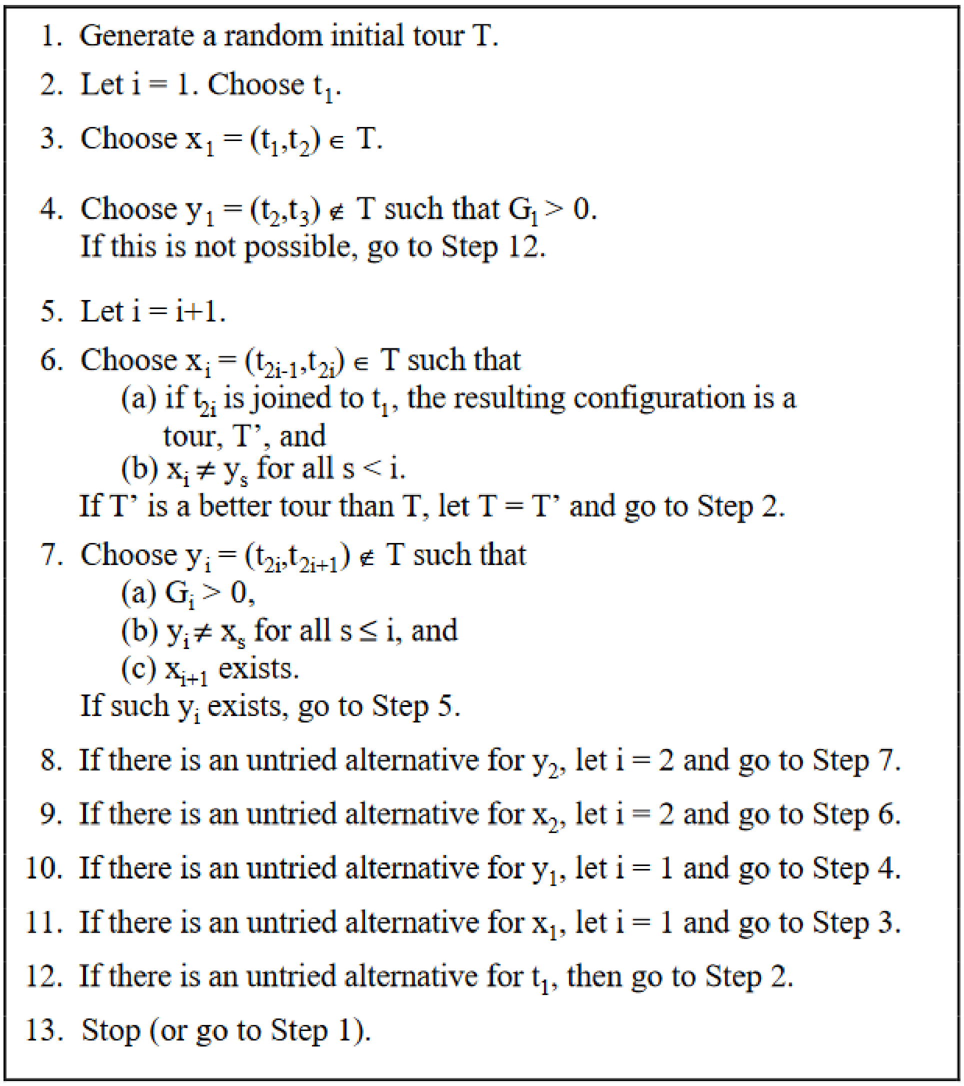
   It is required that the sets X and Y are disjoint. This simplifies coding, reduces running time and gives an effective stop criterion.

The outline of the basic Lin-Kernighan Algorithm can be seen in Figure 2.7.

A bottleneck of the algorithm is the search for links to enter the sets X and Y. In order to increase efficiency, special care therefore should be taken to limit this search. Thus, to limit the search even more *Lin and Kernighan refined the algorithm* by introducing the following rules.

5. The search for a link to enter the tour, $y_i = (t_{2i}, t_{2i+1})$, is limited to the five nearest neighbors to t_{2i}.

6. For $i \geq 4$, no link, x_i, on the tour must be broken if it is a common link of a small number (2-5) of solution tours.



```
1.  Generate a random initial tour T.
2.  Let $i = 1$. Choose $t_1$.
3.  Choose $x_1 = (t_1,t_2) \in T$.
4.  Choose $y_1 = (t_2,t_3) \notin T$ such that $G_1 > 0$.
    If this is not possible, go to Step 12.
5.  Let $i = i+1$.
6.  Choose $x_i = (t_{2i-1},t_{2i}) \in T$ such that
        (a) if $t_{2i}$ is joined to $t_1$, the resulting configuration is a
            tour, T', and
        (b) $x_i \neq y_s$ for all $s < i$.
    If T' is a better tour than T, let T = T' and go to Step 2.
7.  Choose $y_i = (t_{2i},t_{2i+1}) \notin T$ such that
        (a) $G_i > 0$,
        (b) $y_i \neq x_s$ for all $s \leq i$, and
        (c) $x_{i+1}$ exists.
    If such $y_i$ exists, go to Step 5.
8.  If there is an untried alternative for $y_2$, let $i = 2$ and go to Step 7.
9.  If there is an untried alternative for $x_2$, let $i = 2$ and go to Step 6.
10. If there is an untried alternative for $y_1$, let $i = 1$ and go to Step 4.
11. If there is an untried alternative for $x_1$, let $i = 1$ and go to Step 3.
12. If there is an untried alternative for $t_1$, then go to Step 2.
13. Stop (or go to Step 1).
```

Figure 2.7. *The basic Lin-Kernighan Algorithm [5]*

7. The search for improvements is stopped if the current tour is the same as a previous solution tour.

8. When link y_i ($i \geq 2$) is to be chosen, each possible choice is given the priority $c(x_{i+1}) - c(y_i)$.

9. If there are two alternatives for x_4, the one where $c(x_4)$ is highest is chosen.

Lin's and Kernighan's refinements are mostly heuristics rules. 5 through 7 primarily aim to limit the search, while 8 and 9 primarily focus on directing the search.

Lin-Kernighan-Helsgaun Algorithm

A modified and extended version of their algorithm was presented in Helsgaun's work in [5]. This algorithm is a considerable improvement of the original algorithm. The increase in efficiency is primarily achieved by a revision of Lin and Kernighan's heuristic rules for restricting and directing the search. Although their heuristic rules seem natural, a critical analysis can easily show that they suffer from several defects.

A central rule in the original algorithm is the heuristic rule that restricts the inclusion of links in the tour to the five nearest neighbors to a given city (Rule 5). This rule directs the search against short tours and reduces the search effort substantially. However, there is a certain risk that the application of this rule may prevent the optimal solution from being found. If an optimal solution contains one link, which is not connected to the five nearest neighbors of its two end cities, then the algorithm will have difficulties in obtaining the optimum. For example, for a 532-city problem [42] one of the links in the optimal solution is the 22nd nearest neighbor city for one of its end points.

Helsgaun introduced the α-measure for specifying this candidate set, and is found to be much better than using nearest neighbor approach. Firstly, it is important to understand the *1-tree* concept, whose definition follows. *A 1-tree for a graph $G = (N, E)$ is a spanning tree on the node set $N \setminus \{1\}$ combined with two edges from E incident to node 1.* The choice of node 1 as a special node is arbitrary. Note that a 1-tree is not a tree since it contains a cycle. An explanatory example is depicted in Fig. 2.8.

A minimum 1-tree is a 1-tree of minimum length. An optimal tour normally contains between 70 and 80 percent of the edges of a minimum 1-tree. Therefore, minimum 1-trees seem to be well suited as a heuristic measure of 'nearness'. Edges that belong, or 'nearly belong', to a minimum 1-tree, stand a good chance of also belonging to an optimal tour. Conversely, edges that are 'far from' belonging to a minimum 1-tree have a low probability of also belonging to an optimal tour.

This measure of 'nearness' can be defined as follows. *Let T be a minimum 1-tree of length $L(T)$, and let $T^+(i, j)$ denote a minimum 1-tree required to contain the edge (i, j). Then the α-nearness of an edge (i, j) is defined as the quantity $\alpha(i, j) = L(T^+(i, j)) - L(T)$.* Thus, the α-nearness of an edge is the increase of length when a minimum 1-tree is required to contain this edge.

The α-measure can be used to systematically identify those edges that could conceivably be included in an optimal tour, and disregard the remainder. These 'promising edges',

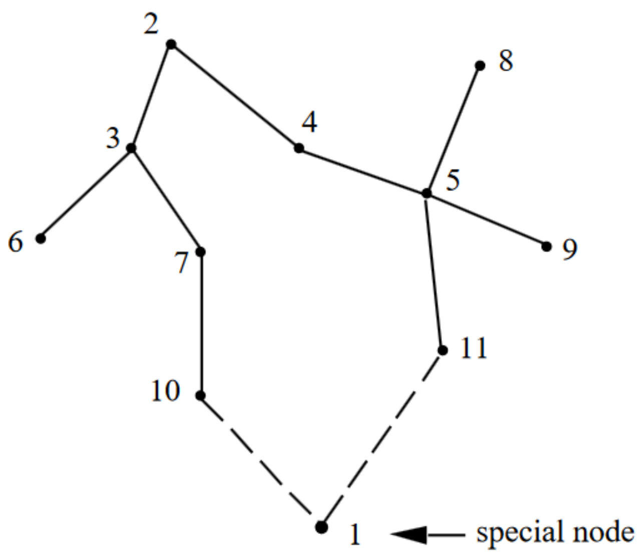


Figure 2.8. *A 1-tree [5]*

called the candidate set, may, for example, consist of the k α-nearest edges incident to each node, and/or those edges having an α-nearness below a specified upper bound.

This approach, enhances a lot the original one, but it would not be of much practical if the computations of those values were too expensive. An algorithm is presented in [5] that computes all α-values, with time complexity being $O(n^2)$ and space complexity $O(n)$.

2.3.4 Simulated Annealing

Simulated annealing, is among the best known local search heuristic algorithms, it performs quite well and is widely applicable.

In condensed matter physics, annealing is known as a thermal process for obtaining low energy states of a solid in a heat bath. The process consists of the following two steps [43]:

- increase the temperature of the heat bath to a maximum value at which the solid melts;
- decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid.

It is known that, if the lowering of the temperature is done sufficiently slowly, the solid can reach thermal equilibrium at each temperature. In the Metropolis algorithm [44] this is achieved by generating a large number of transitions at a given value of the temperature. Thermal equilibrium is characterized by the Boltzmann distribution, which gives the probability of the solid of being in a state / with energy E_i at temperature T, and which is given by

$$\mathbf{P}_T\{\mathbf{X} = i\} = \frac{\exp\left(-E_i/k_B T\right)}{\sum_j \exp\left(-E_j/k_B T\right)},$$

where X is a random variable denoting the current state of the solid and the summation extends over all possible states.

The following equivalences with combinatorial optimization problems occur.

- solutions in the combinatorial optimization problem are equivalent to states of the physical system;
- the cost of a solution is equivalent to the energy of a state.

A control parameter c_i is introduced, which plays the role of the temperature. A fitness function f is also introduced, which measures the performance for the candidate solution. A typical feature of simulated annealing is that, besides accepting improvements in cost, it also accepts deteriorations to a limited extent. In the pseudo-code of algorithm 1, we can see how the algorithm works.

Algorithm 1: *Simulated Annealing [15]*

```
1: procedure SimulatedAnnealing($i_{start}$, $c_0$, $L_0$)
2:     INITIALIZE($i_{start}$, $c_0$, $L_0$);
3:     $k \leftarrow 0$;
4:     $i \leftarrow i_{start}$;
5:     while stopcriterion = FALSE do
6:         for $l \leftarrow 1$ to $L_k$ do
7:             GENERATE($j$ from $S_l$);
8:             if $f(j) \leq f(i)$ then
9:                 $i \leftarrow j$;
10:            else
11:                if $\exp\left(\frac{f(i)-f(j)}{c_k}\right) > \text{random}[0,1)$ then
12:                    $i \leftarrow j$;
13:                end if
14:            end if
15:        end for
16:        $k \leftarrow k+1$;
17:        CALCULATE_LENGTH($L_k$);
18:        CALCULATE_CONTROL($c_k$);
19:    end while
20: end procedure
```

Therefore for the context of Traveling Salesman Problem, a Simulated Annealing algorithm would commence with a random tour and then proceed based on the k-opt logic (eg. 2-opt), with the difference that it also accepts solutions that produce decline in the overall result, with a relatively small probability, by comparing the value of $\exp\left(\frac{f(i)-f(j)}{c_k}\right)$ with a random number on the interval $[0,1)$. It is obvious that the speed of convergence of the algorithm is determined by the choice of the parameters L_k and c_k with k=*0,1,...*, denoting the iteration step of the algorithm.

Other solutions to the TSP include a variety of approaches, that will not be discussed in this work. In addition to those mentioned above, notable methods include the Christofides algorithm, which provides an approximation solution with a guaranteed upper bound, as well as the Ant Colony Optimization (ACO) algorithm, inspired by the foraging behavior of ants, which efficiently explores the solution space to find near-optimal solutions.

2.4 Generalized Traveling Salesman Problem

The Generalized Traveling Salesman Problem (GTSP) is an extension of the classical Traveling Salesman Problem (TSP) and it is among the most researched combinatorial optimization problems due to its theoretical properties, complexity aspects and real-life applications in various areas. The GTSP is one practical extension of the TSP, first introduced by [45], where the set of vertices V is further segmented into n number of groups and it asks to find a minimum-cost route visting at least one vertex from each group before reaching the destination.

GTSP could mathematically be defined as follows [46]: Let $G = (V, E)$ be a graph where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices, $E = \{(v_i, v_j)|i \neq j; v_i, v_j \in V\}$ is the edge set, and $W = \{w_{ij}\}$ is the non-negative cost or weightage defined on E. If E is undirected, then directions become irrelevant, i.e., $(v_i, v_j) = (v_j, v_i)$. Furthermore, V is partitioned into x mutually exclusive and exhaustive groups such that $V^g = \{V_1, V_2, ..., V_x\}$ and $V = V_1 \cup V_2 \cup ... \cup V_x$ with $V_a \cap V_b = \emptyset$ for all $a, \beta = 1, 2, ..., x$ and $a \neq \beta$. It asks to determine the shortest Hamiltonian route that passes through each group at least once (introduced independently by [45], [47]) or exactly once (introduced by [48]). If the matrix W is symmetrical, i.e., $w_{ij} = w_{ji}$ for all $i, j = 1, 2, ..., n$ and $i \neq j$, the problem is prefixed as symmetric; otherwise, it is asymmetric. This results in many vertices from each group left to be visited.

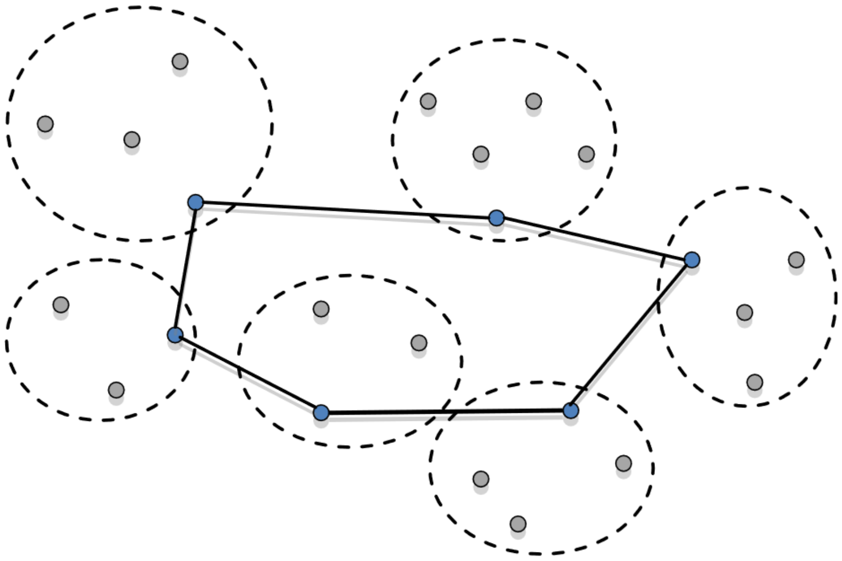


Figure 2.9. *Illustration of the GTSP for an instance with 6 clusters (n=23, m=6). [1]*

The exactly once variant of the *GTSP* is also known as the **Equality Generalized Traveling Salesman Problem**, Equality-GTSP or **E-GTSP**, where the shortest route contains exactly one vertex, i.e., station, from each group in V^g [46]. The E-GTSP is an NP-hard problem [49], as it reduces down to TSP (also NP-hard) whenever individual groups become singleton ($|V_a| = 1, \forall a = 1, 2.., x$). In this context, a *g-tour* is a closed Hamiltonian cycle, that visits exactly one vertex from each group before returning to the starting point.

2.4.1 Overview of GTSP Solution Approaches

In this section, we examine various approaches to solve the GTSP. We categorize these approaches based on the optimization techniques they employ [2], as shown below.

Exact Algorithms

The exact methods are cable of producing the optimal solution for a given optimization problem. Generally, for NP-hard problems, these methods are very time-consuming because of the complexity of the calculations, thereby making them applicable in very limited cases. Therefore, exact approaches are adopted in only a few papers and they perform well only on small problem instances.

[45], [47] and [50] proposed dynamic programming (DP) approaches for solving the GTSP derived from DP methods for the classical TSP, in which a state is defined by the clusters that have been already visited. A major drawback of these DP approaches is that the number of states grows exponentially as the number of clusters increases.

[51] described a branch-and-bound algorithm for solving the GTSP based on the minimal rooted tree as a relaxation. The authors were able to solve instances with up to 13 clusters and 52 vertices optimally.

[48] provided a method based on integer linear programming that can be applied to both Euclidean and non-Euclidean problems, with the only restriction that the distance matrix is symmetric. Their proposed exact method was rather effective.

[52] described a branch-and-cut method that generated optimal solutions for problems with up to 89 clusters and 442 vertices; size still insufficient for real-world applications. This type of algorithms generally requires very large computation times to solve large instances.

Transformation methods

Due to the complexity of generalized combinatorial optimization problems, transforming them into classical combinatorial optimization problems is a convenient solution, although such transformations are usually implying a growth in the size of the problems' instance. Transformation methods of the GTSP into the TSP have been researched in several works.

The first transformation of the GTSP into the TSP was proposed in [53]. A major drawback of this transformation is that the number of vertices of the transformed TSP is more than three times larger than the number of vertices in the corresponding GTSP.

A quite efficient transformation of the asymmetric GTSP into the classical asymmetric TSP is described in [19]. Their transformation was done in two steps and is characterized by the fact that it does not increase the number of vertices but slightly increases the number of arcs.

[54] proposed a different transformation that dropped off the size of the corresponding TSP. In their transformation, a GTSP instance is transformed into an instance of the asymmetric TSP having twice the number of vertices as the original GTSP. Modifying this transformation, [55] described a more efficient transformation in which the number of vertices in the transformed TSP does not surpass the number of vertices in the original GTSP.

Reduction algorithms

An important characteristic of GTSP is that it is not necessary to visit all the vertices of the graph. The GTSP can contain vertices that do not belong to the optimal solution and therefore may be eliminated. An analogous situation is encountered regarding the edges. Thus, reduction algorithms that eliminate either vertices or edges of the underlying graph might be of interest.

Three reduction algorithms that eliminate redundant vertices and edges, while maintaining the value of the optimal solution, can be found in [16]. The first reduction algorithm eliminates the redundant vertices, the second one eliminates the redundant edges, and the third one both. All proposed reduction methods keep the vertices and edges that are in the optimal solution. The proposed combined reduction algorithm has a running time of $O(n^3)$ in the worst case scenario, reduced the size of the instance by 15–20% and lowered the computational time by approximately 45%.

Another reduction method was proposed in [56] . Their pre-processing technique selects from every cluster the closest vertices to the other clusters and removes the vertices that have never been chosen to reduce the solution search space size. The proposed method had very small running times, while the rate of the reduction is up to 98%, very competitive against the reduction algorithms described in the previous paragraph (proposed in [16]).

Approximation algorithms

Approximation algorithms are polynomial time algorithms that produce approximate solutions to NP-hard optimization problems, with demonstrable guarantees on the quality of the solution. The design and analysis of approximation algorithms include a mathematical proof confirming the quality of the generated solutions in the worst case, distinguishing them from heuristic approaches, which find reasonably good solutions, but do not provide any clear indication about the quality of the solutions.

The most prominent research in this direction can be found in the works of [57], [58] that exploits the classical Christofides approximation algorithm [59]. Also, in later works [60] and [61].

Heuristic algorithms

When it comes to solving problems faster and traditional methods are too slow, or when an exact solution cannot be yield by means of a traditional method and a suboptimal solution needs to be identified, heuristic algorithms come into play. The main distinction between these classes of techniques is that exact methods ensure that a solution is optimal, while heuristic methods may provide good quality solutions but without any assurance of optimality.

A composite heuristic for solving the GTSP, called Generalized Initialization, Insertion and Improvement, denoted by GI3, which is an extension of the heuristic algorithm presented in [62] for the classical TSP, was introduced in [63].

A random-key genetic algorithm (GA) for solving the GTSP was proposed in [64]. The advantage of their approach was that solutions produced by the crossover or the mutation operators are feasible solutions of the GTSP.

A memetic algorithm in which a genetic algorithm is combined with local search techniques is also provided in [65]. The main contribution of the authors was the originality of the crossover operator which relies on large neighborhood search.

Helsgaun [1] combined the efficient transformation of the asymmetric GTSP into the classical asymmetric TSP developed by [19] with the powerful Lin-Kernighan-Helsgaun (LKH) TSP solver to solve the transformed GTSP instances. In this way, he was able to improve the quality of the solutions for the GTSP_LIB [52] instances over the best existing algorithms at that time.

An efficient solution approach for solving the GTSP that relies on adaptive large neighborhood search, called GLNS, was presented in [17]. Their algorithm removes and inserts vertices over and over again into the generalized tour. Its main characteristic is the introduction of a general insertion mechanism that includes as specific cases the well-known nearest, farthest, and random insertion mechanisms.

A basic iterated local search (Basic ILS) and a refined version of the algorithm (Refined ILS) was provided in 2022 in [18].

2.4.2 Evaluation Datasets

The existing datasets used for testing the performance of different solution approaches for the GTSP belong to four libraries, as analyzed in [2].

1. GTSP_LIB was proposed in [52] organizes vertices into clusters based on their proximity to each other. It iteratively selects $m = \lceil n/5 \rceil$ cluster centers, ensuring that each chosen center maximizes its distance from the nearest previously selected center. The remaining vertices are assigned to the cluster whose center is closest to them. This way of clustering the vertices simulates the geographical regions. GTSP_LIB contains 88 symmetric and asymmetric instances with up to 1084 vertices and 217 clusters. The instances were acquired from the Reinelt's TSP_LIB [66].

2. BAF_LIB was proposed in [67], by Bontoux as part of his PhD thesis. The instances from this library were derived as well from Reinelt's TSP_LIB [66]. A main feature of generating the clusters is that there are no geographical regions, like before, and the clusters are generated pseudo-randomly. BAF_LIB contains 56 instances which are symmetric with up to 1084 vertices and 217 clusters.

3. MOM_LIB was introduced in [68] initially for the case of the clustered TSP, then for the GTSP and the clustered shortest-path tree problem [69]. MOM_LIB includes six kinds of Euclidean instances which were generated using distinctive algorithms [68]. There were small instances with vertices ranging from 30 to 120, and clusters ranging from 2 to 42. There were large instances with vertices ranging from 108 to 3000, and partitioned into clusters ranging from 4 to 200.

4. LARGE_LIB was proposed in [1] and has 44 very large, symmetric instances with the number of clusters ranging from 10 to 17.180 and the number of vertices ranging from 1.000 to 85.900. These instances were derived from TSP_LIB and the National TSP benchmark library [70]. The clusters were obtained using Fischetti's clustering procedure [52].

2.4.3 Comparative Analysis of Existing Algorithms

The best performing state of the art algorithms for solving the GTSP that have been published in the literature are:

- the memetic algorithm (MA) described by Gutin & Karapetyan (2010) [16],
- the Lin-Kernighan-Helsgaun (LKH) solution approach provided by Helsgaun (2015) [1],
- the large neighborhood search (LNS) algorithm described by Smith & Imeson (2017) [17],
- a basic iterated local search (Basic ILS) and a refined version of the algorithm (Refined ILS) provided by Schmidt & Irnich (2022) [18].

Pop et al in their work [2], thoroughly describe the comparisons between these state of the art solvers for GTSP. We summarize this analysis in the following paragraphs.

The box plots of the success rates and the average percentage errors obtained on different GTSP libraries based on the processed results are shown in Fig.2.10 - Fig.2.13.

The *success rate* of an algorithm, when solving an instance, represents the percentage of runs at which it found the Best Known Solution (BKS). The *percentage error* e that occurs when an instance is solved by an algorithm is given by the following formula:

$$e = 100 \cdot \frac{\text{foundSolution} \ - BKS}{BKS} \%$$

In conclusion, based on the above, we observe that the LNS algorithm, also called GLNS, described in [17] by Smith & Imeson showed the best overall performance. Yet, considering the GTSP_LIB dataset, and only there, Helsgaun's GLKH method [1] showed slightly superior performance.

Both solutions are thoroughly presented in the two following subsections.

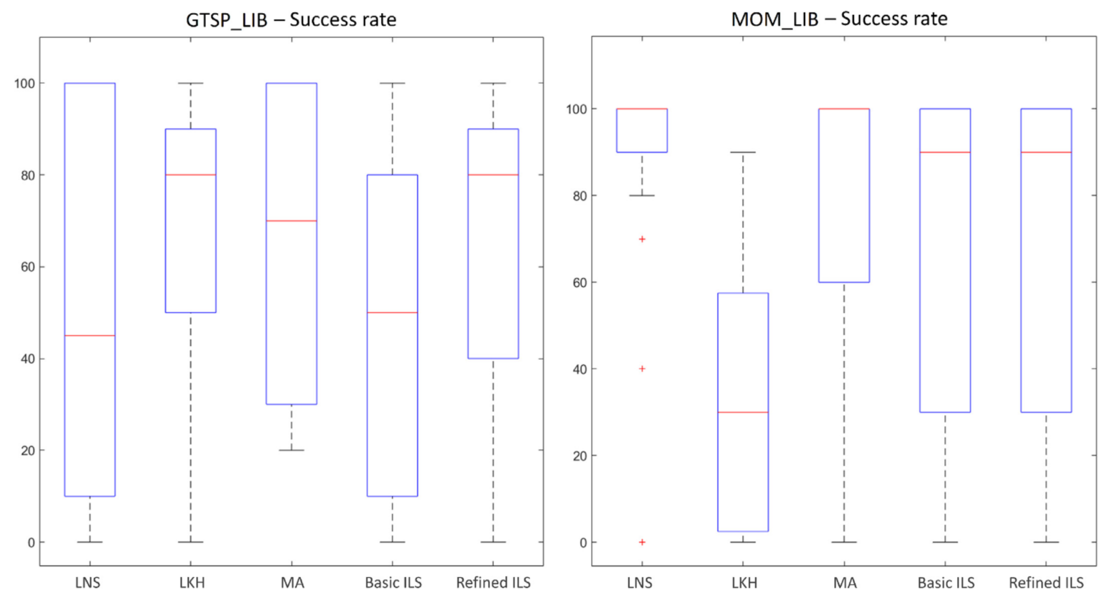


Figure 2.10. *Box plots of the success rate for five algorithms on the GTSP_LIB and MOM_LIB instances [2]*

.

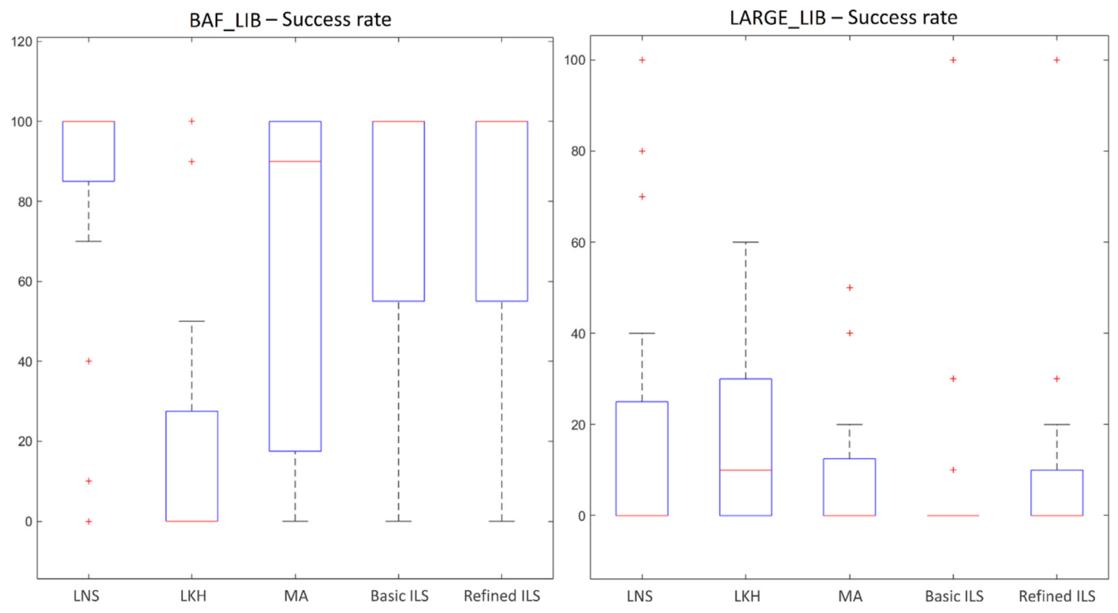


Figure 2.11. *Box plots of the success rate for five algorithms on the BAF_LIB and LARGE_LIB instances [2]*

.

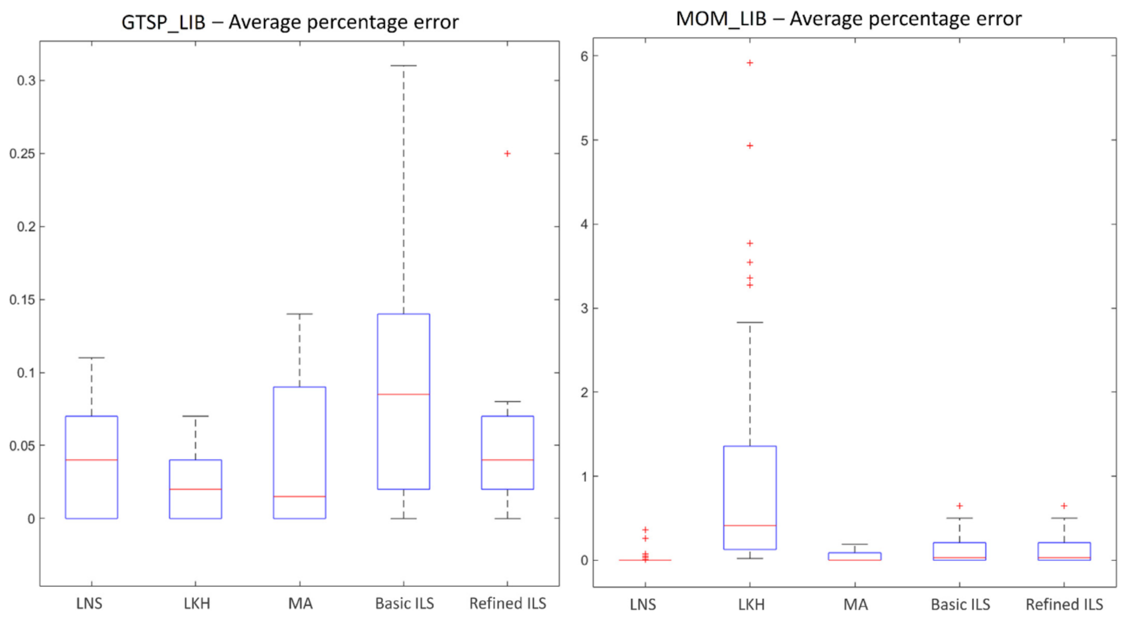


Figure 2.12. *Box plots of the average percentage error for five algorithms on the GTSP_-LIB and MOM_LIB instances [2]*

.

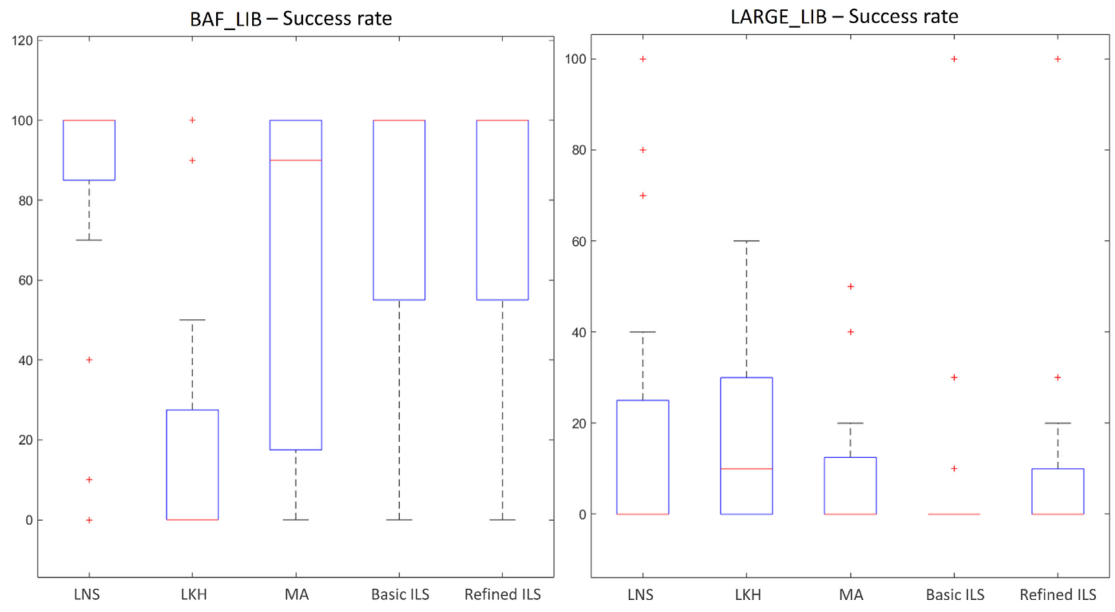


Figure 2.13. *Box plots of the average percentage error for five algorithms on the BAF_-LIB and LARGE_LIB instances [2]*

.

2.4.4 GLKH Solution

This solution transforms the E-GTSP problem to a TSP and employs the LKH algorithm to solve the instance. It is well known that any E-GTSP instance can be transformed into an asymmetric TSP instance containing the same number of vertices. This work takes advantage of the approach introduced by Noon & Bean in [19], who state that he transformation can be described as follows, where V' and c' denote the vertex set and cost matrix of the transformed instance:

1. V' is equal to V.
2. Create an arbitrary directed cycle of the vertices within each cluster and define ${c'}_{ij} = 0$ when v_i and v_j belong to the same cluster and v_j succeeds v_i in the cycle.
3. When v_i and v_j belong to different clusters, define $c'_{ij} = c_{kj} + M$, where v_k is the vertex that succeeds v_i in a cycle, and M is a sufficiently large constant. It suffices that M is larger than the sum of the n largest costs.
4. Otherwise, define ${c'}_{ij} = 2M$.

This transformation works because once a cluster is entered at a vertex v_i, an optimal TSP tour always visits all other vertices of the cluster before moving to the next cluster. The optimal TSP tour must have zero cost inside the cluster and the inter-cluster edges must be exactly m . Thus, the cost of the g-tour for the E-GTSP is the cost of the TSP tour minus mM. The g-tour can be extracted by selecting the first vertex from each cluster in the TSP tour.

The aforementioned transformation allows one to solve E-GTSP instances using an asymmetric TSP solver. However, in the past this approach has had very little application, due to the unusual structure of the produced TSP, which is hard to handle for many existing TSP solvers. Since a near-optimal TSP solution may correspond to an infeasible E-GTSP solution, heuristic TSP solvers are often considered inappropriate [71, 72]. In [1], it is shown that this need not be the case if the heuristic TSP solver LKH is used.

The E-GTSP solver based on LKH, also referred to as GLKH, follows a structured procedure to solve the problem efficiently. First, it reads the E-GTSP instance and then transforms it into an asymmetric TSP instance. Next, it writes the transformed TSP instance to a problem file and specifies suitable parameter values in a separate parameter file. Once the input files are prepared, the solver executes LKH using these files. After obtaining the solution to the TSP instance, GLKH extracts the g-tour from the TSP solution tour. Finally, it performs post-optimization of the g-tour, refining the solution further to improve its quality.

2.4.5 GLNS Solution

Smith and Imeson in their work in 2017 [17] proposed the GLNS solution for the Equality Generalized Traveling Salesman Problem (GTSP). This solver operates under the general framework of adaptive large neighborhood search (ALNS) [73]. The idea is relatively simple. We begin with an initial solution and then iteratively destroy and repair until a termination condition is reached. In the ALNS framework there are two types of heuristics, one for insertions and one for removals. Weights are associated with each one and are being updated during the procedure. This mechanism of altering the weights results to the name *adaptive*.

The GLNS algorithm can be seen right below.

Algorithm 2: *GLNS(G, P_V) [17]*

```
1: **Input:** A GTSP instance $(G, P_V)$.    ▷ G: Graph, $P_V$: partition of vertices V into sets
2: **Output:** A GTSP tour on $G$.
3: for $i = 1$ to $num\_trials$ do
4:     $T \leftarrow initial\_tour(G, P_V)$
5:     $T_{\text{best},i} \leftarrow T$
6:     repeat
7:         Select a removal heuristic $R$ and insertion heuristic $I$ using the selection weights
8:         Select the number of vertices to remove, $N_r$, uniformly randomly from $\{1, ..., N_{\max}\}$
9:         Create a copy of $T$ called $T_{\text{new}}$
10:        Remove $N_r$ vertices from $T_{\text{new}}$ using $R$
11:        For each of the $N_r$ sets not visited by $T_{new}$, using $I$ insert a new vertex into $T_{\text{new}}$
12:        Locally re-optimize $T_{\text{new}}$
13:        if $w(T_{\text{new}}) < w(T_{\text{best},i})$ then
14:            $T_{\text{best},i} \leftarrow T_{\text{new}}$
15:        end if
16:        if accept($T_{\text{new}}$, $T$) then
17:            $T \leftarrow T_{\text{new}}$
18:            Record improvement made by $R$ and $I$
19:        end if
20:    until stop criterion is met
21:    Update selection weights based on improvements of each heuristic over trial
22: end for
23: return tour $T_{\text{best},i}$ that attains $\min_i w(T_{\text{best},i})$
```

The algorithm starts with an initial (random) tour. It repeatedly performs removals and insertions, updating at the end the scores of each removal and insertion heuristic based on their success. It randomly selects the number of vertices to be removed, uniformly in the range 1 to N_{max}. Based on a simulated annealing criterion (line 16 of algorithm) it accepts or declines the modified tour T_{new}. Then certain stopping criteria are employed. The first phase is an initial descent, which stops after a fixed number of non-improving iterations. The second consists of several warm restarts, each one beginning with the best solution found so far, yet with a lower simulated temperature. Each restart also ends after a certain number of non-improving steps.

Four insertion heuristics were proposed [52] for iteratively constructing the GTSP tour:

- *Nearest insertion* picks the set V_i that contains a vertex v at minimum distance to a vertex on the partial tour T. This means that we choose the set V_i as follows:

$$\underset{V_i \in P_V \setminus P_T}{\operatorname{argmin}} \min_{u \in V_T} \operatorname{dist}(V_i, u).$$

- *Farthest insertion* picks the set V_i whose closest vertex to a vertex on the partial tour T is maximum. This means:

$$\underset{V_i \in P_V \setminus P_T}{\operatorname{argmax}} \min_{u \in V_T} \operatorname{dist}(V_i, u).$$

- *Random insertion* picks uniformly randomly from $P_V \setminus P_T$, a set V_i.
- *Cheapest insertion* picks the set V_i that contains the vertex v that minimizes the insertion cost, as follows:

$$\underset{V_i \in P_V \setminus P_T}{\operatorname{argmin}} \min_{v \in V_i, (x,y) \in E_T} \{w(x,v) + w(v,y) - w(x,y)\}.$$

The above can be unified into a single insertion heuristic and thus all of them can be selected based on certain probabilistic criteria (thoroughly discussed in [17]).

A removal heuristic framework is described in the algorithm below.

Consider a partial tour T containing $l \in \{m - N_r + 1, \ldots, m\}$ vertices, where for simplicity of notation, the vertices are numbered such that $V_T = \{1, \ldots, v_l\}$ and $E_T = \{(v_1, v_2), (v_2, v_3), \ldots, (v_l, v_1)\}$. Then, for a fixed parameter λ and a set of distances r_j for each $v_j \in V_T$, the general removal framework is specified in the following algorithm.

Algorithm 3: *Removal heuristic framework for a given λ and distance metric r_j.*

```
1: Input: A partial tour $T = (V_T, E_T)$, $\lambda \in [0, \infty)$, and values $r_j$ for each $v_j \in V_T$
2: Output: A new tour with one vertex removed from $V_T$
3: Randomly select $k \in \{1, ..., l\}$ according to the unnormalized probability mass function
   $[\lambda, \lambda, ..., \lambda^{l-1}]$.
4: Pick the vertex $v \in V_T$ with the kth smallest value $r_j$.
5: Remove $v$ from $V_T$
6: Remove $(v_{j-1}, v_i)$ and $(v_j, v_{j+1})$ from $E_T$ and add $(v_{j-1}, v_{j+1})$ to $E_T$.
7: return $T$
```

The weights for both the insertions heuristics employed and the removal ones are being adapted/updated after each iteration. The score for each iteration is calculated as follows:

$$\text{score } = \max\left\{\frac{w(T) - w\left(T_{\text{new }}\right)}{w(T)}, 0\right\}$$

This score gives the fractional improvement in tour cost, where the max ensures that we do not penalize a heuristic when it increases the tour cost. At the end of a trial, the overall score for a heuristic is given by the sum of its scores, divided by the number of times it was used (i.e. the average score). We then update the weight of each heuristic as ϵ times the previous weight plus $1 - \epsilon$ times the average score on the trial.

Local tour optimization techniques are also employed. Two of them are the *Re-Optimize Vertex in Each Set (Re-Opt)*, and *Move-Opt*. The first one re-optimizes the vertex in each set, keeping the ordering of the sets fixed, while the latter attempts to optimize the ordering of the sets. More information can be retrieved from the original work in [17].

2.5 Available Technologies

There are several available technologies that would help us handle and solve our problem. Our work primarily is based on the following:

- C++, which is a high-level, general-purpose programming language created by Danish computer scientist Bjarne Stroustrup in 1985,
- Julia language, which is a dynamic programming language primarily designed for numerical analysis and computational science. It offers high-performance capabilities and is increasingly popular among researchers and scientists, and
- ROS, which stands for Robot Operating System. It is an open-source robotics middleware suite. It also offers tools and libraries that help developers create complex and robust robot applications.

Chapter 3

GTSP Planner Implementation

3.1 Introduction

As already discussed in the introduction of this thesis, we want to solve a quite complex multi-objective optimization problem. A user is going to select a set of elements that need to be automatically inspected. Our programs should propose an optimal sequence of waypoints in order to maximize the visibility of the targets for the inspection task, while simultaneously proposing a low-cost route.

Many ideas can be stated to solve this optimization problem. We propose a solution that breaks down the problem into to smaller ones and solve each one effectively and efficiently. It's main components are the selection of possible waypoints (arranged in clusters), the solution to a path-planning problem between these points to generate a cost matrix and the solution of a Generalized Traveling Salesman Problem based on the elements that the user wants to be inspected.

3.2 Data Format

The data with which we work with consists of information retrieved after a quite informative walk of the robot in the high voltage facility's outdoor environment. The robot was tele-operated and covered a satisfying distance, yet not all the environment extensively. The route it followed can be seen in Figure 3.1. In the following paragraphs the format of the data available is thoroughly presented. Later on, based on this information we propose our methodology/solver.

Elevation data is presented in Figures 3.1, 3.2, and 3.3. In Figure 3.1 the elevation is depicted in the super-safe edition, which exclusively refers to the map of the route that the robot followed during the data-gathering walk. Figure 3.2 illustrates the entire mapping of the environment which as expected is not as accurate since the robot did not cover the entire distance. This mapping was generated using algorithms based on the data collected by the robot. Therefore, precise measurements of each region are crucial for accurately creating the map. Finally, Figure 3.3 provides a side-view perspective of the elevation data.

The elevation data combined with the roughness of the terrain, the slope and proper calculations resulted in the *traversability map*, shown in Fig. 3.6, 3.7. The data we work with in this work is binary (0 and 1); zero values represent non-traversable areas, while ones represent traversable ones. In the aforementioned figures, ones are shown by the puprle cells, while zeros by the black. A next step is to transform the traversability map to account for a range of values, from non traversable cells and low traversible points to higher ones. The algorithms we use work for both approaches.

With the proper analysis and implementation of our team, based on ray-casting algorithms, *visibility maps* were generated. Some of them can be seen in Figures 3.4 and 3.5, each one referring to a different element. We can easily detect zones where the visbility is high for the certain element, red indicating the best points. Based on the above implementation each element might also has a different visibility map for each side of it, depending on which exact part of it we want to inspect.

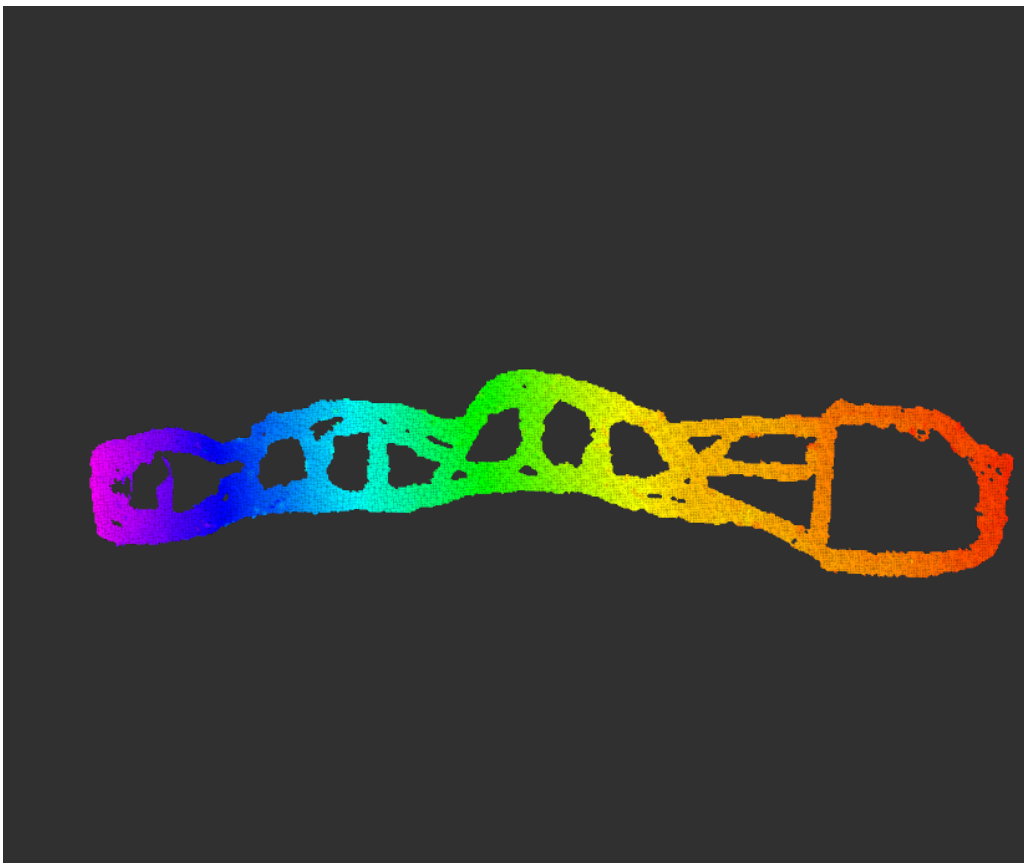


Figure 3.1. *Top-view of the route that the robot followed on its data-gathering walk. The colors indicate the elevation data. In the left-side there there is low elevation, while in the right-side is higher.*

3.3 Methodology

In order to solve the problem we developed a method, which is described in detail in flowchart of Fig. 3.8. The method can be divided into two separate procedures. The one runs offline, once, and its the data preprocessing step. The second part functions on demand, triggered whenever a user requests a plan.

In flowchart of Fig. 3.8 the first two brackets, named "Calculate interesting points"

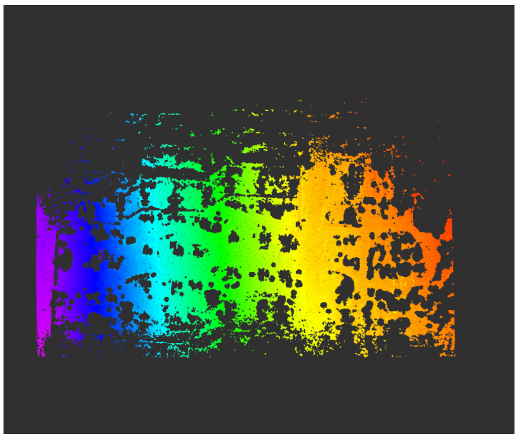


Figure 3.2. *The whole map of the environment based on the data the robot gathered and analyzed. Color shows elevation data.*

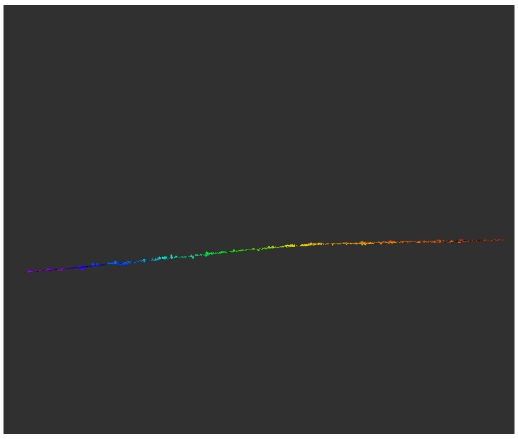


Figure 3.3. *Elevation data of the environment, side-view. Left-side shows low elevation, while right-side much higher. Color indicates elevation data.*

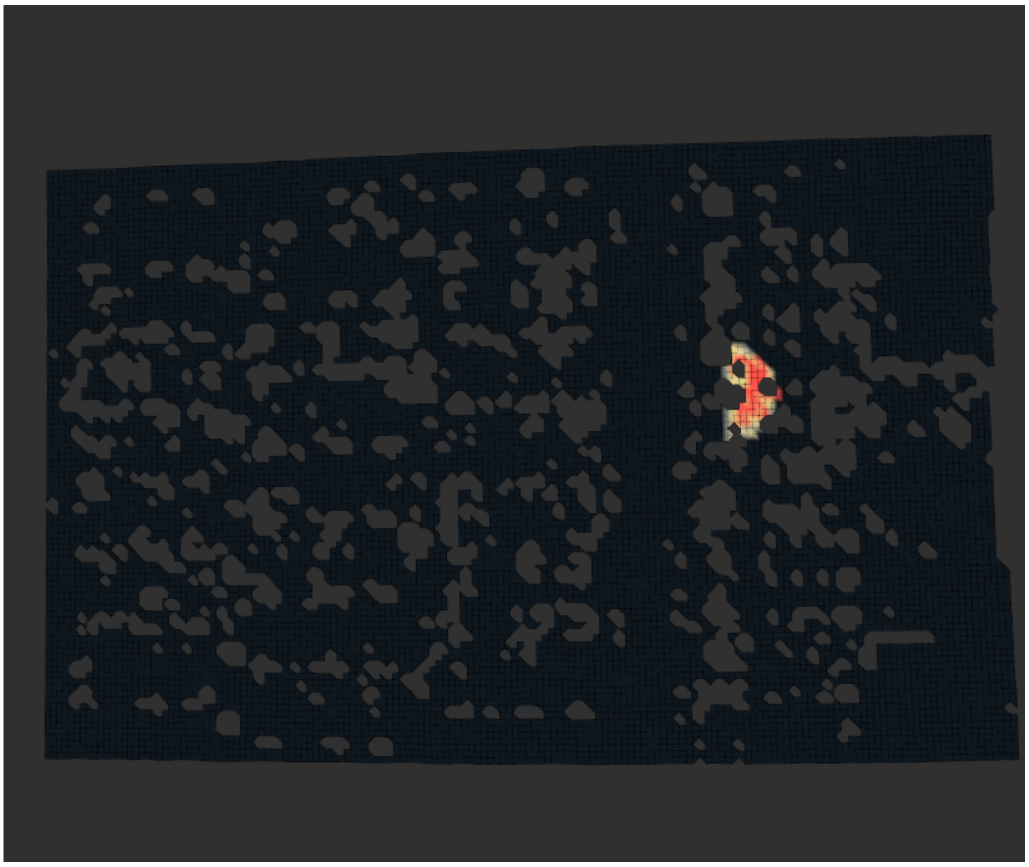


Figure 3.4. *Representation of the visibility data for an element. We can see the whole map, as well as the part with high visibility coloured. Red indicates better visibility score.*

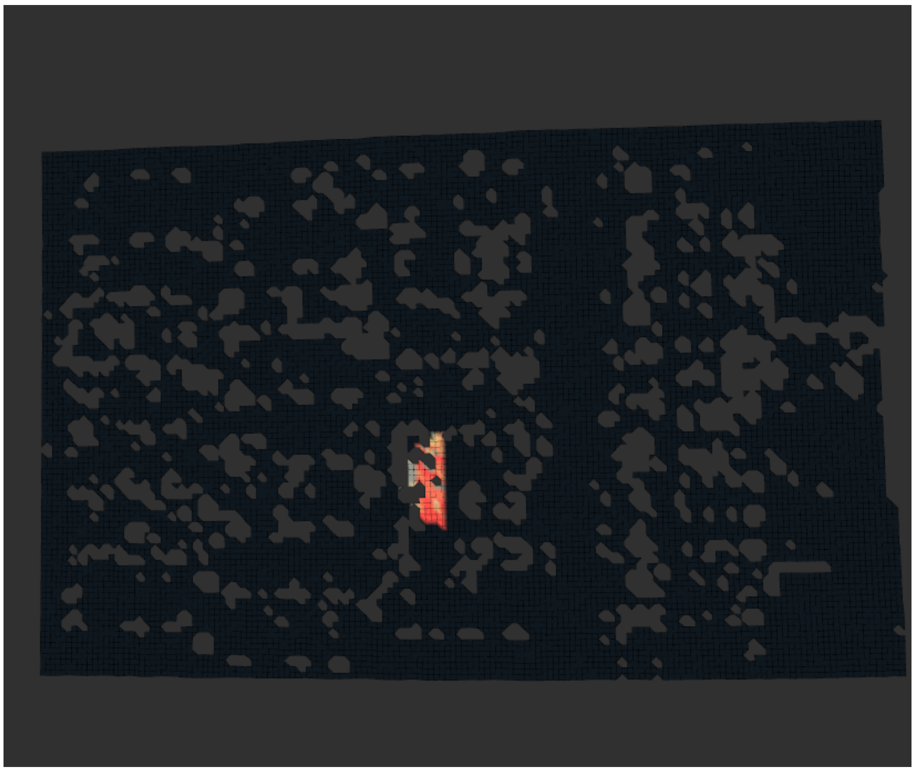


Figure 3.5. *Representation of the visibility data for an element. We can see the whole map, as well as the part with high visibility coloured. Red indicates better visibility score.*

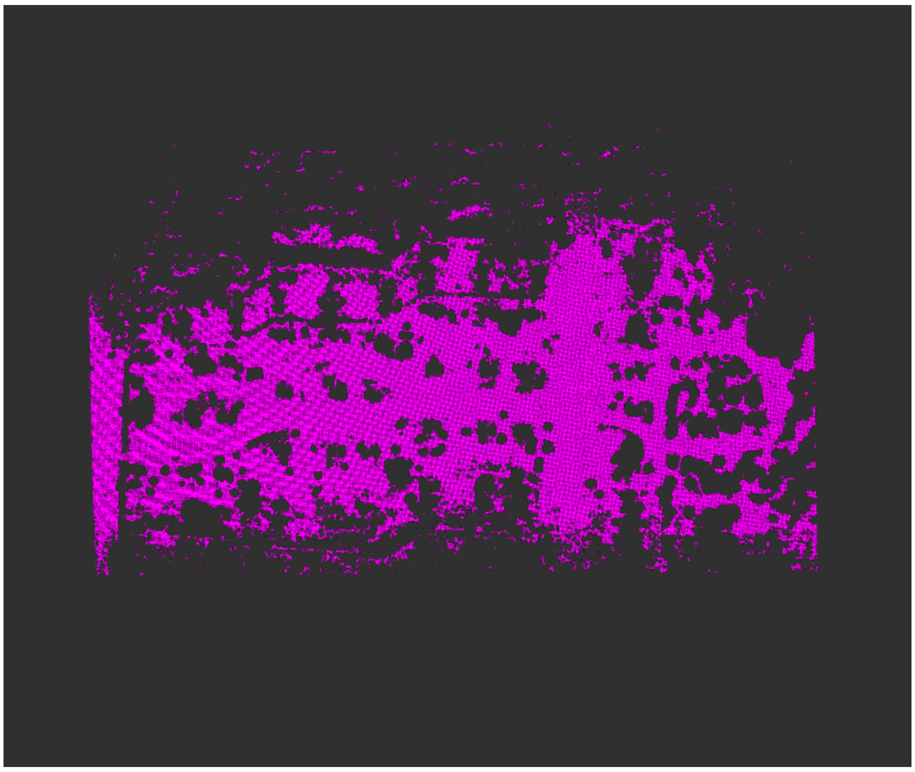


Figure 3.6. *Representation of the traversability data for the whole map.*

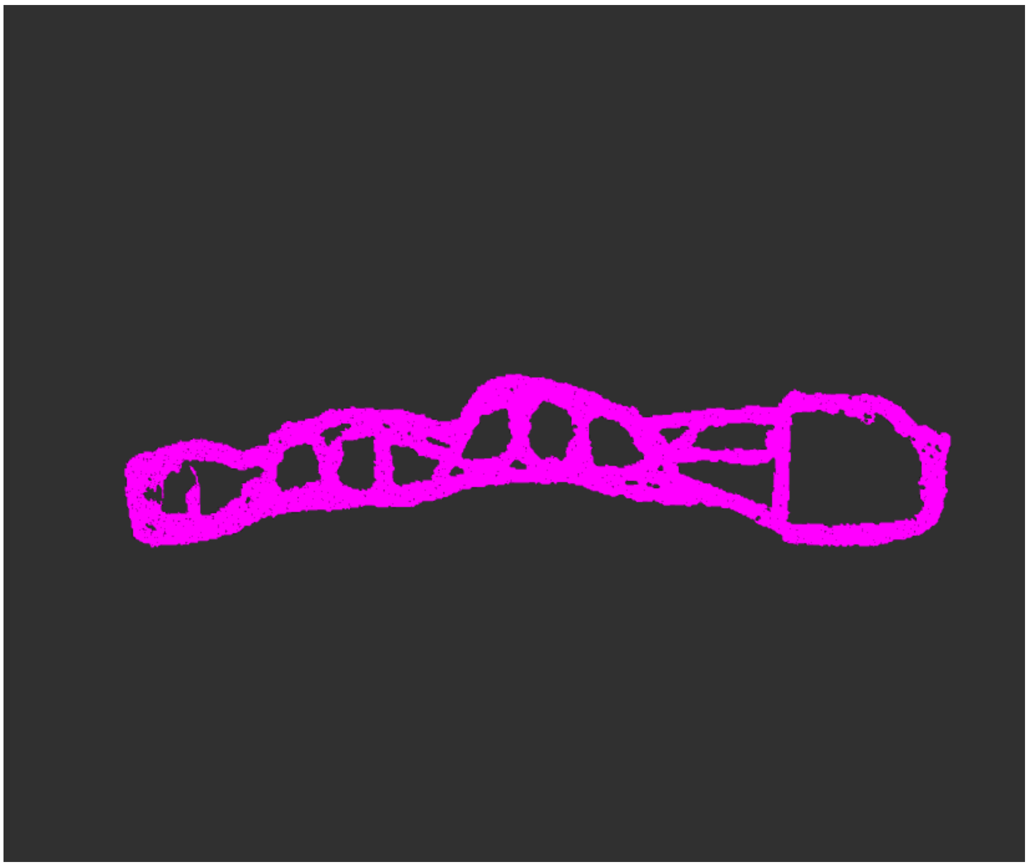


Figure 3.7. *Representation of the traversability data for the super-safe map.*

and "Compute Offline Distances" respectively refer to the preprocessing step, while the two last brackets refer to solving the exact problem instance.

In the following subsections we delve into how these procedures exactly work.

3.3.1 Calculate Interesting Points

The *1st step* is to calculate the interesting points, where interesting refers to points with high visibility scores for every possible element that might need inspection in the future. Each element is analyzed separately based on its visibility data. A cluster of points (where cluster size ≥ 1) is generated, with points indicating high visibility of the exact target.

In Fig 3.9, a simple visual explanation is shown to demonstrate the need of point clustering. For each element, there might be more than one point with good visibility score. We need to consider many of them as visiting potential waypoints for our planning and optimize later in the path-planning steps. Needless to say that the points within the cluster must have some distance between them so as to provide adequate differentiation. In Fig 3.9, the x's represent the extracted waypoints and the rectangles the elements to be inspected.

In order to determine these interesting points for each element, we employ a *greedy algorithm* based on the information provided from the visibility maps (eg. Fig. 3.4-3.5).

To begin, we preprocess the data by assigning the average score of a 0.5m radius region

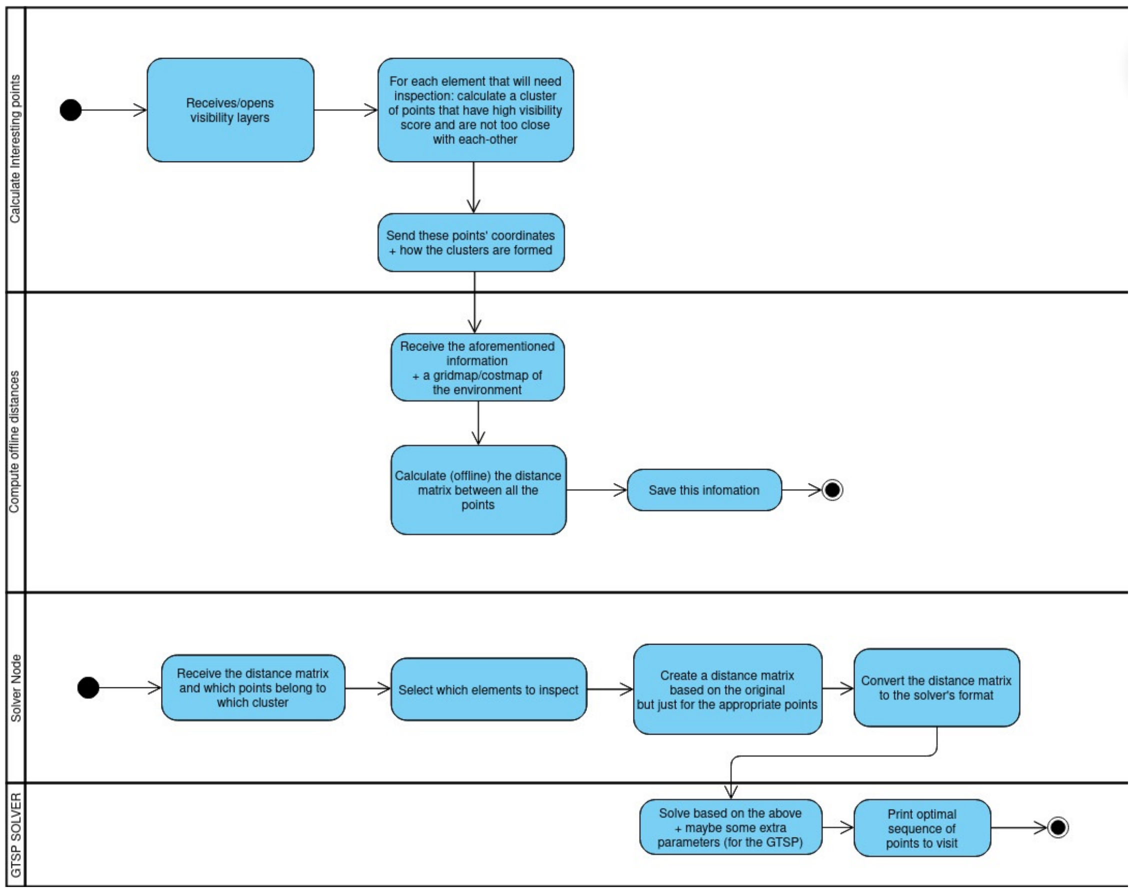

Calculate Interesting points
Receives/opens visibility layers
For each element that will need inspection: calculate a cluster of points that have high visibility score and are not too close with each-other
Send these points' coordinates + how the clusters are formed
Compute offline distances
Receive the aforementioned information + a gridmap/costmap of the environment
Calculate (offline) the distance matrix between all the points
Save this infomation
Solver Node
Receive the distance matrix and which points belong to which cluster
Select which elements to inspect
Create a distance matrix based on the original but just for the appropriate points
Convert the distance matrix to the solver's format
GTSP SOLVER
Solve based on the above + maybe some extra parameters (for the GTSP)
Print optimal sequence of points to visit

Figure 3.8. *A flowchart to better explain the created procedure.*

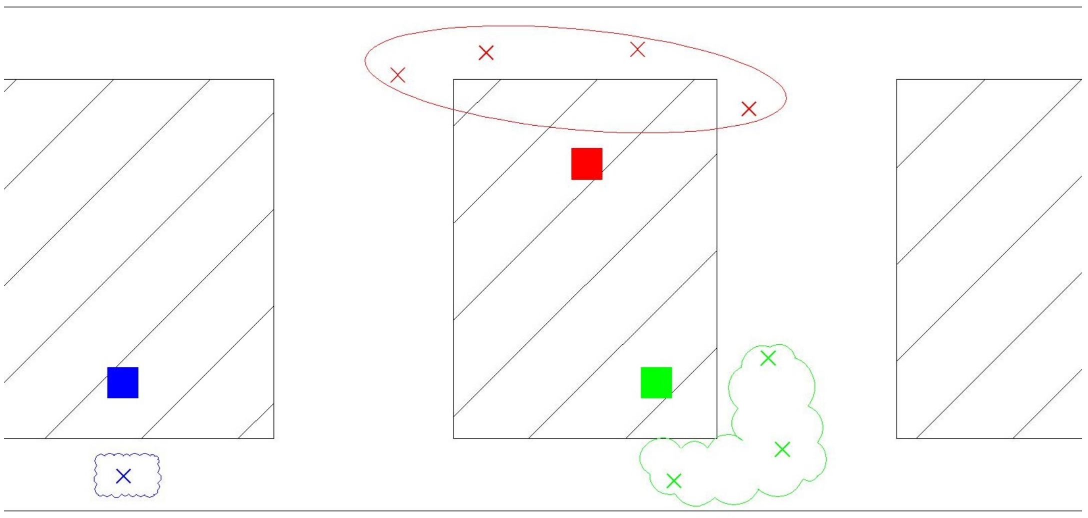


Figure 3.9. *One possible scenario to better explain how the clustering works. Rectangles are the elements to be inspected, while the x's represent viewpoints and form clusters respectively.*

to each cell of the gridmap. This approach prioritizes visiting regions with high scores rather than individual points.

Next, we sort each point based on its score using the C-language standard library sort function, which employs the *introsort algorithm* for sorting, which is a is a hybrid sorting algorithm that provides both fast average performance and optimal worst-case performance.

Then the *greedy algorithm* takes place. It works as follows. It firstly chooses the point with the best score, produced by the sorting. If multiple points share the same score, it randomly selects one among them. Then it proceeds to select the next best point that is not within a 1.5m radius of any previously chosen interesting points. This process continues until either five points have been selected or no additional points with scores above a certain threshold are found.

In the Figure 3.10 we can see as an example a clusters of three points (yellow markers), depicted above the visibility map in reference to a certain element.

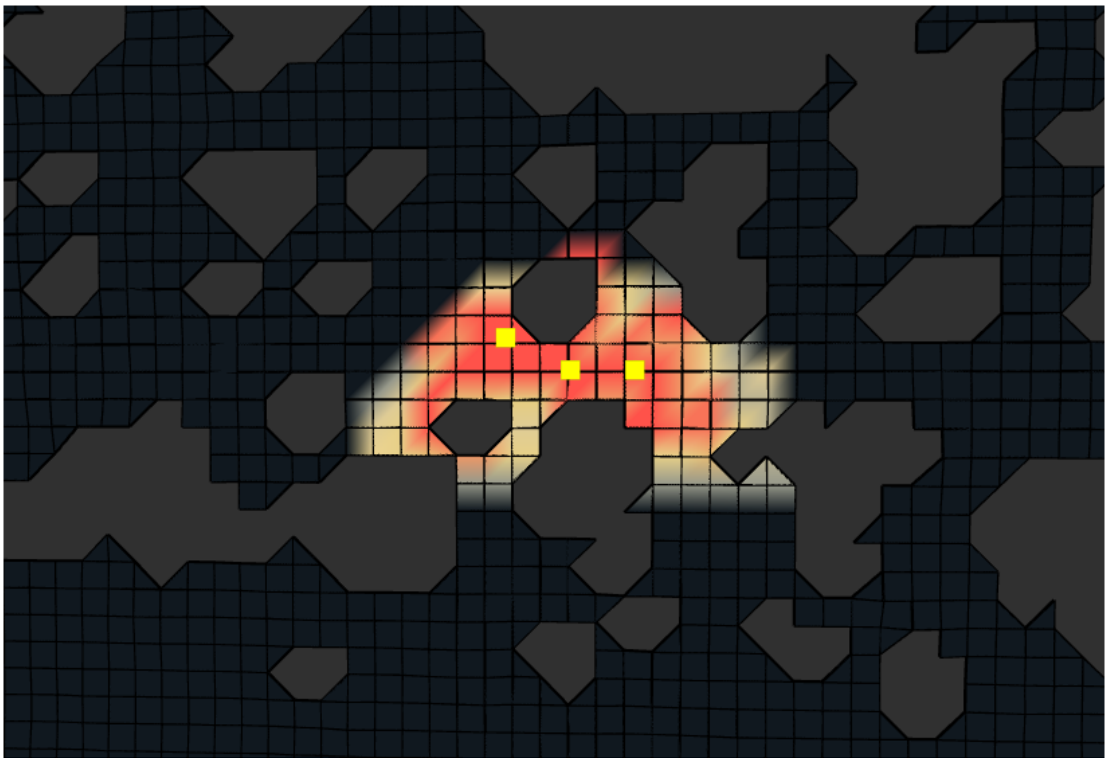


Figure 3.10. *An example of a cluster of points extracted, visualized with yellow markers in the visibility map. The points appear to be into red zones, with some distance between them.*

3.3.2 Compute Distances

The *2nd step* of our procedure immediately follows the calculation of the interesting/high scoring visibility points and the appropriate clustering, described before.

This offline pre-processing step occurs only once, and its primary objective is to prepare the necessary information, *compute the distances* between the points, for extracting optimal plans later on. Optimality is not the primary concern at this stage, as it runs once and offline. Yet, accurate calculations are essential due to the fact that the will be repeatedly exploited in the next steps and determine the accuracy of the solutions.

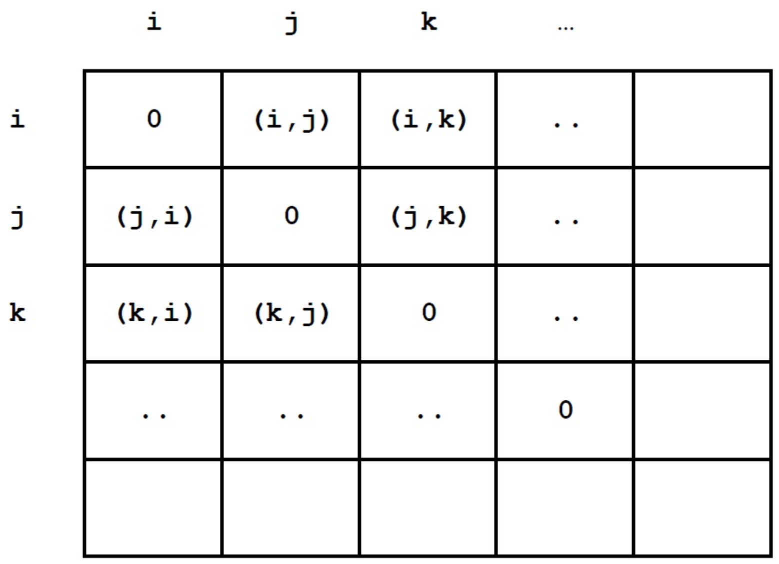
So, we have a list of interesting points extracted from the previous step. Between all these pair of points we want to calculate the exact cost based on the traversability map's data.

The outcome we wish for, with the following priority, is

1. the highest possible accuracy of the calculations (i.e. exact cost),
2. lowest possible complexity for the above.

To achieve the outcome we want with relatively low complexity, we employ the *A* algorithm*, with euclidean heuristic function, so as to produce heuristic proposals always lower or equal to the actual distance to the goals and thus be admissible. We run for each pair of points. The advantages of the selection of this algorithm are also emphasized in Chapter 2.

Therefore we create a full matrix, as shown in Figure 3.11, where each point refers to a traversal cost (e.g. (i,j) refers to the cost from *point i* to *point j*). We perform such a solution because we want to adapt our traversability maps for binary to range values, creating thus a costmap based on elevation and other terrain information that are going to create asymmetries.



	i	j	k	...	
i	0	(i,j)	(i,k)	..	
j	(j,i)	0	(j,k)	..	
k	(k,i)	(k,j)	0	..	
	..	..	..	0	

Figure 3.11. *Full distance matrix format. (a,b) refers to the traversal cost from node a to node b.*

3.3.3 Solve An Instance of the Problem

This step is not executed off-line, is the one *executed on demand.* When the user specifies the elements for the autonomous inspection, certain steps are taken.

With proper retrieval of the information analyzed in the previous steps we have insight about the possible points we are going to visit, how the clusters are formed and the distances between all these points. In reality we have a big distance matrix possessing all the possible pairs' distances, the clusters we want to visit, and which points are located into these clusters. A proper slicing takes place in this original matrix and a new one is retrieved consisted only of points that might be waypoints in the specific problem's solution.

After we have created the matrix we transform it to the appropriate file format for the GTSP Solvers to solve, as shown right below.

Listing 3.1: GTSP File Format for Solvers

```
NAME: GTSP_format_explanation
TYPE: GTSP
DIMENSION: 6
GTSP_SETS: 3
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
0 200 2310 6052 404 10912
240 0 2166 5895 634 10719
2326 2195 0 3900 2724 8801
6195 6031 3978 0 6599 4888
434 639 2723 6457 0 11356
10869 10598 8697 4725 11217 0
GTSP_SET_SECTION
1 1 3 -1
2 2 4 5 -1
3 6 -1
EOF
```

The above file describes a problem and will be used by a GTSP solver to be solved. It describes a relatively small problem consisted of 6 total points and 3 clusters/sets (1st one consisted of points numbered 1 and 3, 2nd of {2,4,5} and 3rd one consisted just of one point numbered 6). The EDGE_WEIGHT_SECTION depicts the retrieved matrix just for the appropriate points. Notice that the matrix might also be non symmetrical for datasets that are not binary and account for a range of values. For example a lower cost is assigned if a traversal happens to be downhill, than uphill etc. For our analysis, due to the fact that we work with a binary traversability map, the matrices are symmetric.

Solvers give the optimal sequence of points. GLKH's solution for example is "3, 6, 4". The (x, y) coordinates of these points are also avalaible, so our output are the coordinated of the waypoints the robot is going to visit with the proper sequence.

Both GLKH and GLNS Solvers were employed and tested. Our dataset is quite small to extract conclusions on which is better for it by testing it. We need to determine based on characteristics of it.

A thorough analysis of the different state-of-the-art solvers and a benchmarking of which one performs better in each of the available datasets is presented thoroughly in Chapter 2. A first thought would be to categorize our dataset's format as more similar to GTSP_LIB. Yet although the clusters are formed like geograpichal regions, there are certain scenaria, like the one shown in Fig. 3.12, where the points might not be located in such an orchestrated way. We concluded that *GLNS Solver* is going to be our solver, which showed the best overall performance in the aforementioned analysis.

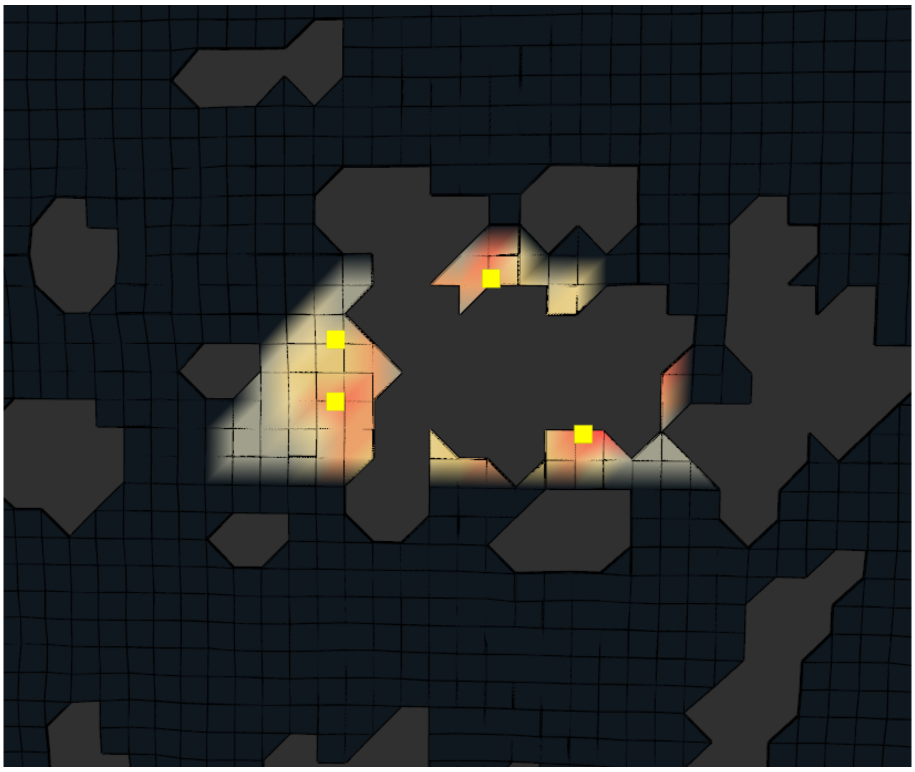


Figure 3.12. *Cluster example to illustrate a possible scenario*

3.4 Implementation

In order to implement the above, we used the following technologies and tools.

- We decided to implement our code in C++, in order to result in relatively high-performing executable files with ROS integration.

- We used the ROS framework for our implementation and exploited some functions from the libaries *gridmap, costmap* and *navfn.* They mainly helped in handling the map's data, using the format both provided by gridmap and costmap. Then we could implement our greedy algorithm for the computation of the interesting points, and experiment with different planners for creating the distance matrices.

- We then used the GTSP Solvers GLKH and GLNS, provided by the authors of these papers (widely available in the Internet). The latter needs Julia language properly configured in our machines for it to run.

- Last but not least, we used the *RViz ROS* tool which helped us in the visualization of each part of our contribution. Some output images from these visualizations are the ones shown in our analysis in Chapters 3 and 4. It helped a lot debug mistakes of our code, as well as extract useful conclusions.

The code of our implementation will be made available on the GitHub profile of @andvatistas for broader access.

Chapter 4

Experimental Results

In this chapter, we present the experimental results of the proposed GTSP planner, for the inspection task problem we want to solve in an outdoor high-voltage facility environment. The data for our experiments was captured from a high voltage facility in Pallini, Greece.

4.1 Setup

In the context of our specific solution a ROBOTNIK mobile robot, model named SUMMIT XL [74] is being used for the navigation in the high-voltage facility. The robot is equipped with a LiFePo4 15Ah@48V battery, a DualShock 4 PS4 controller for tele-operation (if needed for testing things), a Jetway JNF797-Q370 Intel® 8th Generation Core™ i7 as CPU, 250GB SSD, 8GB RAM, encoders in wheels, "VectorNav VN-100T-CR" Inertial measurement unit (IMU), "Orbbec Astra S" RGB Camera, 3D Laser Robosense RS LIDAR 16, GPS u-blox C099-F9P-1 and 4G TELTONIKA RUTX11 router. Figures 4.1 and 4.2 depict the robot in a close-up photo, and in the outdoor environment.

The algorithms of our contribution run in our computers (or on the user's one). The optimal sequence of the points is then being given to the robot to execute the task. Yet, it is useful and important to have mind the actual configuration.

4.2 Experimental Data

As already discussed in Chapter 3, a quite informative tele-operated data gathering walk of our robot, originally took place in the high-voltage facility's outdoor environment. Based on the information received and the proper analysis of it, elevation maps, traversability maps and visibility ones were created. These exact maps retrieved, are the ones also presented as indicative examples in Figures of Chapter 3, in the "Data Format" section, Section 3.2.

Additionally, information from the robot's sensors properly analyzed resulted in the creation of the point-cloud seen in Figures 4.3 and 4.4, where each point represents a single point of the environment. We can easily detect from this point of view several electric poles of the high voltage facility, one next to the other, referring to poles as seen in Figure 1.2 for example.

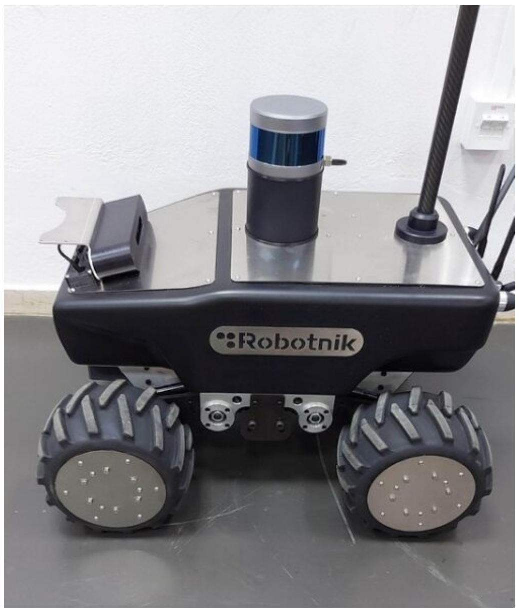


Figure 4.1. *The mobile robot we work with*

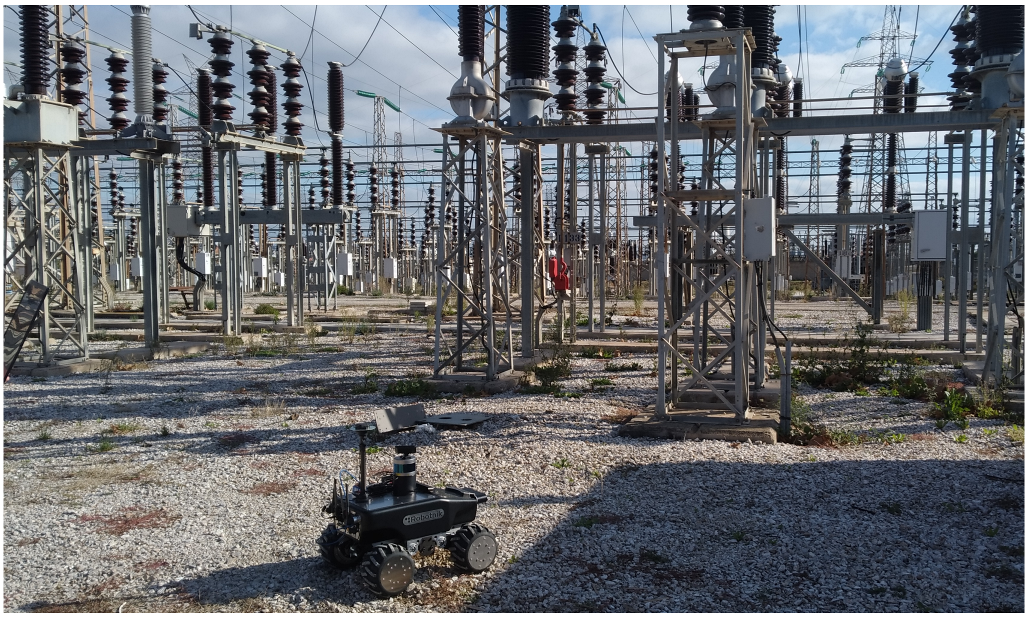


Figure 4.2. *Our mobile robot in the high voltage facility*

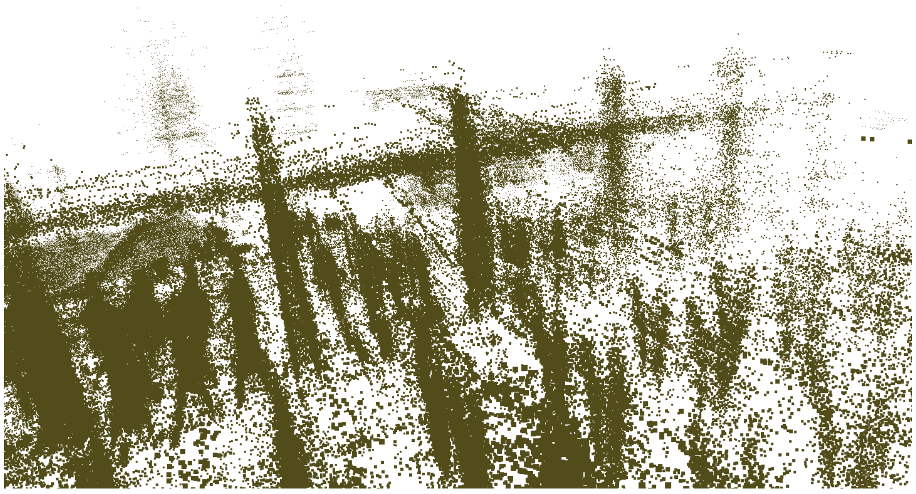


Figure 4.3. *Point cloud of part of the environment. We can easily detect some electric poles, one next to the other.*

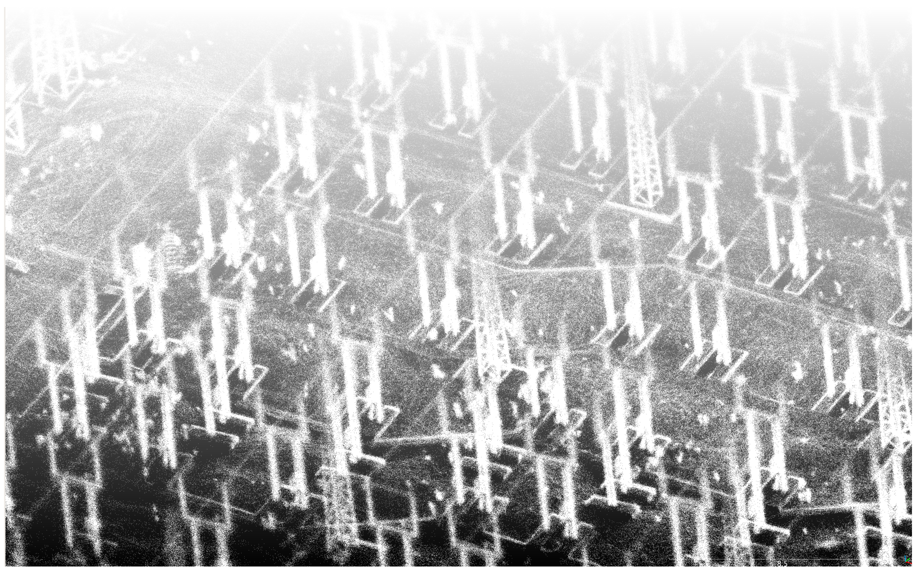


Figure 4.4. *Alternative visualization for the point cloud of part of the environment. We can easily detect some electric poles, one next to the other.*

Also, a representation of the elevation map, combined with the above 3d information is presented in Figure 4.5. We can easily detect with black the electric poles and the other related elements. Simultaneously the colormap indicates the values of the elevation in each cell. High elevation values are represented with red colors, while bluer ones represent lower values.

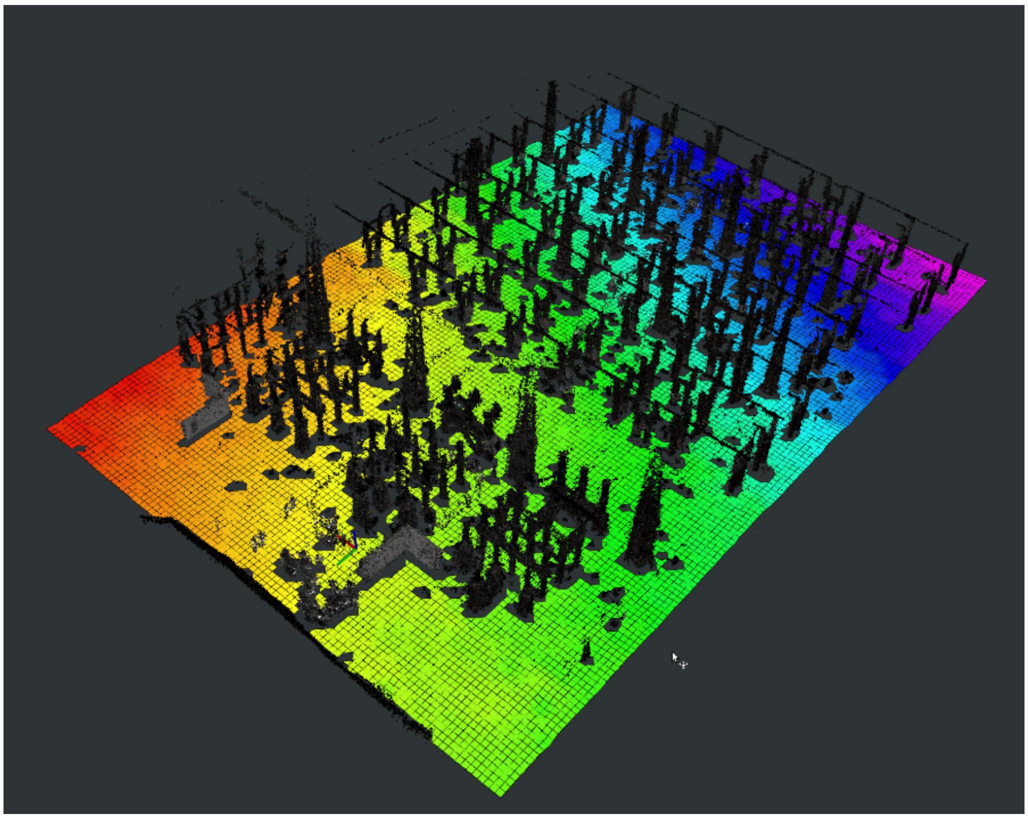


Figure 4.5. *Elevation map combined with the 3d representation of the points in the outdoor high-voltage facility.*

4.3 Results Analysis

As thoroughly already discussed, the result of our planner is the optimal sequence of point coordinates that the robot should visit to optimize its multi-target inspection task. Based on the selected elements for inspection and the environment's related data, our programs operate according to the proposed flowchart and recommend the optimal sequence of points to be visited.

Except for this waypoints' sequence, we can additionally visualize the extracted paths, that our global planner calculated, as shown in Fig. 4.6. This way we can gain insight about the high-level plan of the route that the robot is going to follow. With yellow markers we can see the interesting points, while the green line connects these points based on the global path-planning proposal. Red and black cells indicate traversable areas and non-traversable respectively. As one would expect it only visits one point from each cluster and moves through traversable cells.

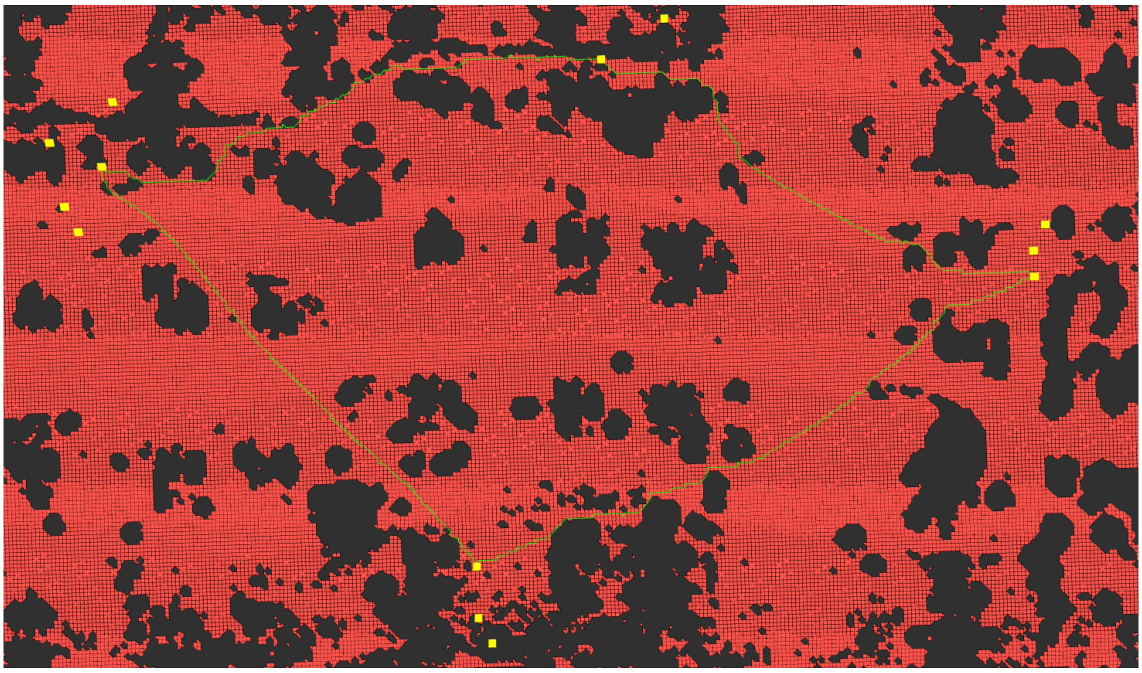


Figure 4.6. *Example of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path.*

Our dataset is really small compared to the instances of the libraries that the GTSP Solvers were originally tested. Our selected solver can extract the optimal routes in fractions of a second for our cases (5-30 elements). Given that we refer to an NP-hard task, a benchmarking here is out of the scope of this thesis. A qualitative analysis was conducted across various scenarios, such as the one depicted in Fig. 4.6. This analysis primarily served to confirm that our algorithms functioned as intended and helped us in debugging during the development phase.

Therefore an extensive qualitative analysis was conducted. In each figure from the ones shown below we solve the problem of the inspection task of 4 different elements. The difference between Figures 4.7-4.11 lies in the starting point.

The starting points can be seen with the blue markers. Four clusters respectively appear and are consisted of the points visualized as yellow markers. Also, the traversability map is shown in red (red equals traversable, black non-traversable areas), and the point cloud of the 3d map's representation in brown. With green line we can see the proposed optimal path.

These experiments confirm that our procedure functions as intended. The running times are remarkably low, while simultaneously the extracted paths for each example are optimal. We can easily detect that each time, a different waypoint for each cluster might be chosen. For instance, in the first two scenarios depicted in Fig.4.7 and in Fig.4.8, concerning the upper right cluster of yellow interesting points, a different waypoint was finally selected as one can easily distinguish.

Our next step for enhancing our results could be an inflation around the objects and generally non-traversable areas of the map, and/or an application of morphological filters. As a result, the robot will be guided through even safer areas, as sometimes it seems to

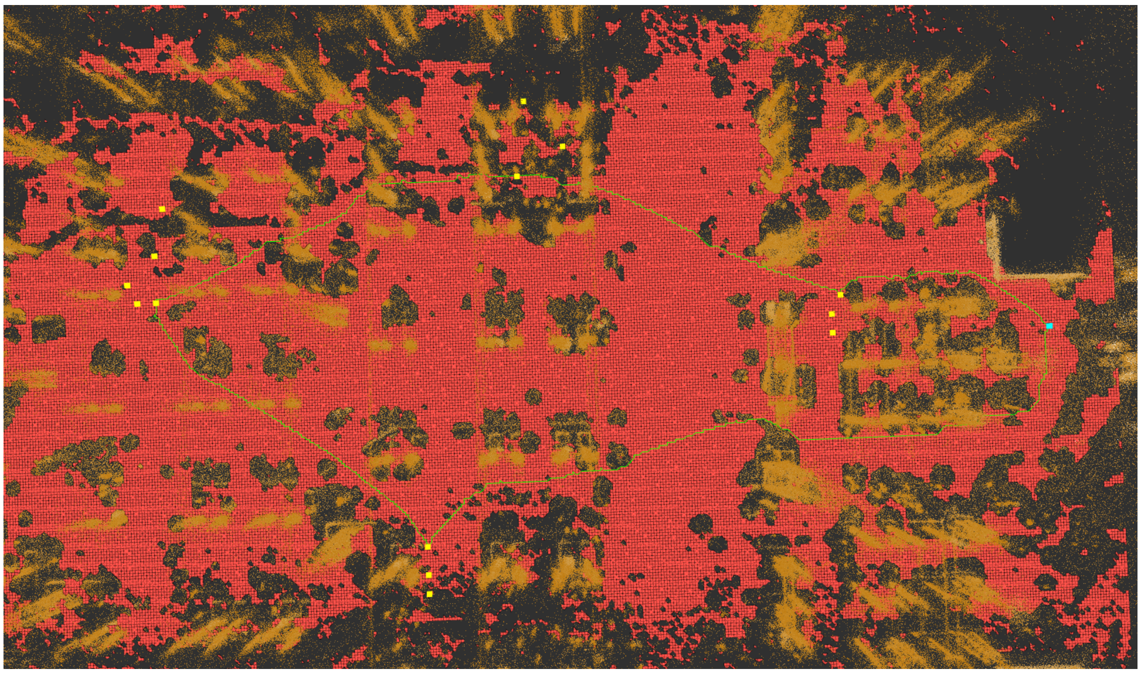


Figure 4.7. *Example 1 of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path. Blue marker indicates the starting point, different each time.*

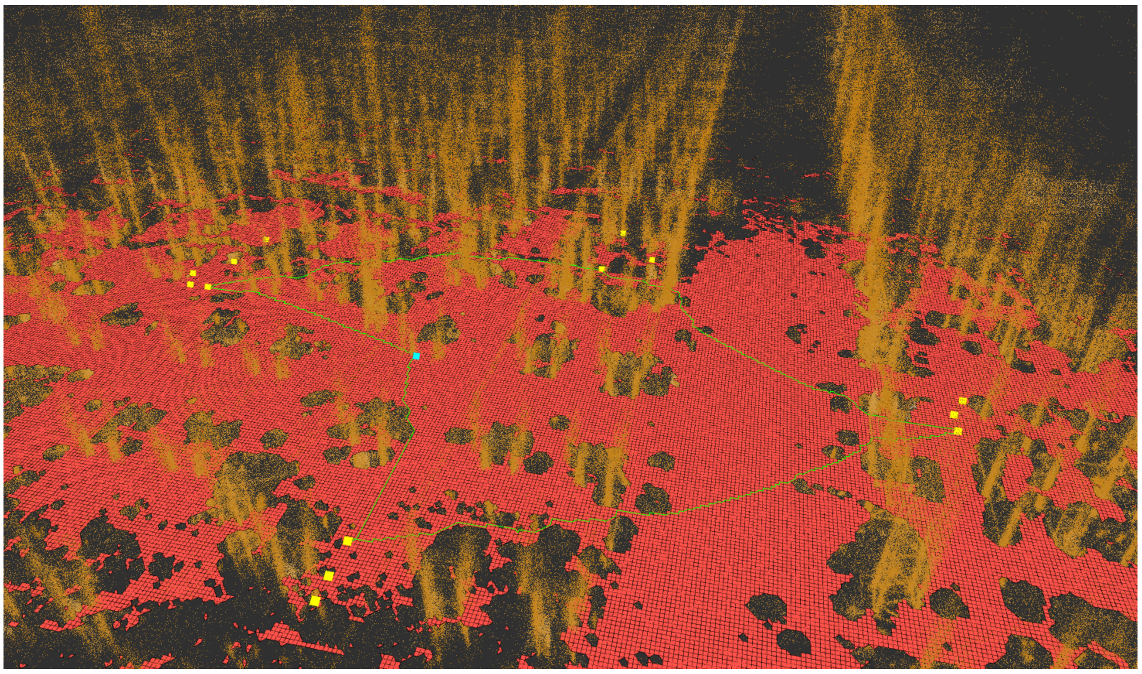


Figure 4.8. *Example 2 of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path. Blue marker indicates the starting point, different each time.*

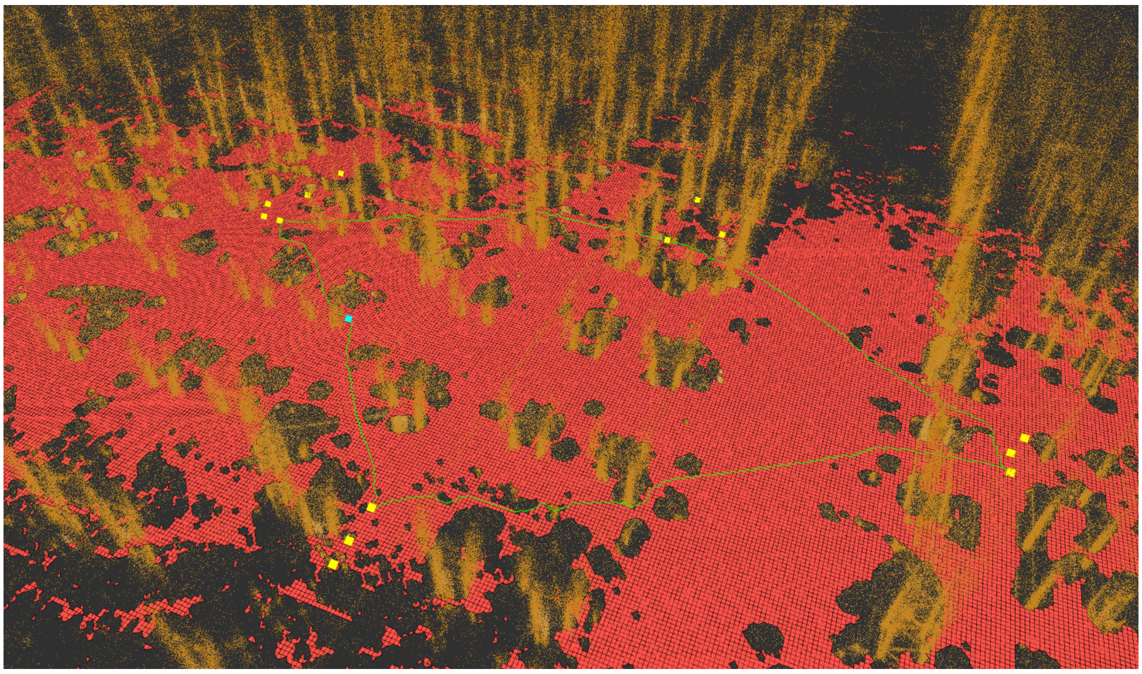


Figure 4.9. *Example 3 of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path. Blue marker indicates the starting point, different each time.*

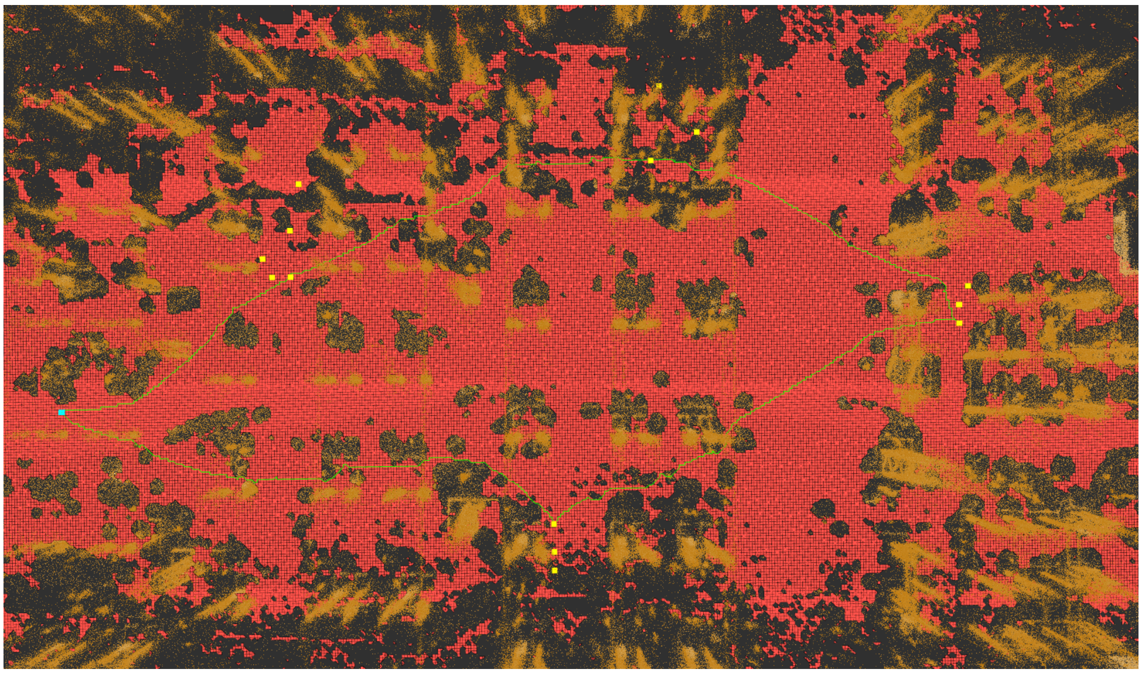


Figure 4.10. *Example 4 of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path. Blue marker indicates the starting point, different each time.*

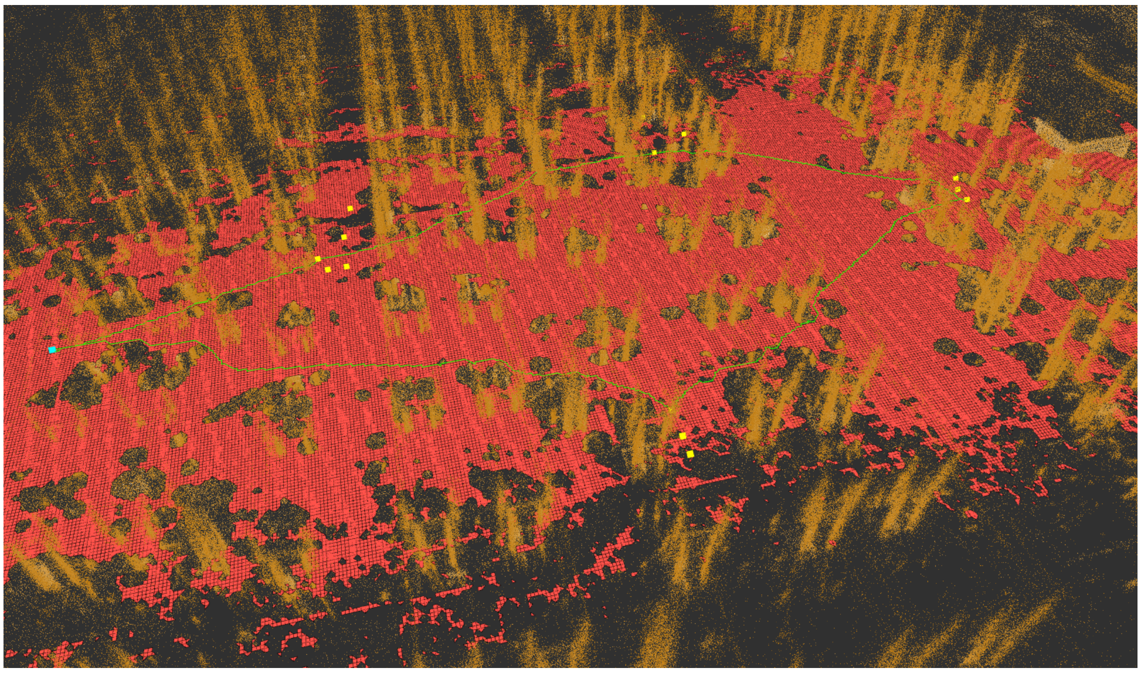


Figure 4.11. *Example 5 of optimal tour extracted for the inspection of 4 elements. Yellow markers show the interesting points, while the green line shows roughly the proposed path. Blue marker indicates the starting point, different each time.*

suggest paths that pass through quite narrow passages that may produce danger.

Chapter 5

Conclusions

5.1 Brief Summary of Thesis Contributions

In this thesis we addressed with a mutli-objective optimization problem for an inspection task of a mobile robot in an outdoor environment. We broke the problem down to seperate subproblems and solved each one optimally.

In Chapter 2 we conducted a comprehensive examination of the existing knowledge within the literature, offering an in-depth analysis. The chapter commenced with an exploration of key terminology and fundamental concepts essential for comprehending subsequent discussions. Our focus then shifted to the path-planning problem, distinguishing between online and offline aspects, with an emphasis on the most widely employed offline methodologies; specifically the Dijkstra algorithm and the A* algorithm. We described these algorithms in depth, provided pseucode and discussed their pros and cons. Then, we discussed the Traveling Salesman Problem and described some prominent approaches, important for developing a fundamental understanding. This foundational knowledge served as a crucial cornerstone for the subsequent exploration of the Generalized Traveling Salesman Problem (GTSP). Our investigation of the GTSP consisted of its mathematical formulation, an extensive overview of GTSP solutions, and a benchmark evaluation of the state of the art algorithms. We then presented some key technologies and tools, used for applications such as ours.

We developed an approach characterized by two distinct stages. The one performed offline for the calculation of interesting/high-scoring visible points and for the cost computation for the traversal between them. The second one performed on demand which deals with the exact problem and proposes the optimal sequence of waypoints to be visited.

After a thorough analysis we employ a greedy algorithm for determining the interesting points, A* for determining the cost matrix and GLNS Solver for solving the GTSP.

Our approach can also be used in several other applications. Needless to say, it can be used in underwater and aerial inspection tasks. Besides, it can be adapted for a range of other purposes with suitable adjustments. For instance, it could be employed to come up with efficient plans for autonomous robotic operations, such as search and rescue missions (covering larger areas by visiting optimal waypoints) and robotic construction applications (for example in welding). Additionally, it could tackle challenges in various other domains where GTSP finds applications (ranging from PCB creation/testing and computer networks to tourism and sightseeing).

5.2 Future Research Directions

The future development of this work could explore several promising directions.

One direction of research involves refining our method for identifying potential interesting points. By enhancing the algorithms and techniques used in this process, we can improve the accuracy and effectiveness of our point selection. A more sophisticated system for determining interesting points could result in clusters with a greater number of points and with higher total average score. Thus leading to more efficient and effective solutions.

Another intriguing possibility is adapting our solution to leverage a fleet of mobile robots. This approach has the potential to significantly enhance the scalability of our system. By coordinating multiple robots to work together, we can cover larger areas more quickly and handle more complex environments. The development of algorithms and protocols for robot collaboration also presents a really exciting research challenge overall.

Bibliography

[1] K. Helsgaun, “Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm,” *Mathematical Programming Computation*, vol. 7, no. 3, pp. 269–287, 09 2015. [Online]. Available: https://doi.org/10.1007/s12532-015-0080-8

[2] P. C. Pop, O. Cosma, C. Sabo, and C. P. Sitar, “A comprehensive survey on the generalized traveling salesman problem,” *European Journal of Operational Research*, vol. 314, no. 3, pp. 819–835, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221723005581

[3] R. Gould, *Graph theory.* Courier Corporation, 2012.

[4] P. Fankhauser and M. Hutter, *A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation.* Cham: Springer International Publishing, 2016, pp. 99–120. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_5

[5] K. Helsgaun, S. o. D. D. A. Roskilde Universitetscenter. Institut for Geografi, and R. universitetscenter. Datalogisk afdeling, *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*, ser. Datalogiske skrifter. Roskilde Universitetscenter, Department of Computer Science, 1998. [Online]. Available: https://books.google.gr/books?id=uexlnQEACAAJ

[6] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Oper. Res.*, vol. 21, pp. 498–516, 1973. [Online]. Available: https://api.semanticscholar.org/CorpusID:33245458

[7] J.-C. Latombe, *Introduction and Overview.* Boston, MA: Springer US, 1991, pp. 1–57. [Online]. Available: https://doi.org/10.1007/978-1-4615-4022-9_1

[8] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, May 2005.

[9] M. Kazemi, K. Gupta, and M. Mehrandezh, *Path-Planning for Visual Servoing: A Review and Issues.* London: Springer London, 2010, pp. 189–207. [Online]. Available: https://doi.org/10.1007/978-1-84996-089-2_11

[10] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn, “Path planning and obstacle avoidance for autonomous mobile robots: A review,” in *Knowledge-Based Intelligent Information*

and Engineering Systems, B. Gabrys, R. J. Howlett, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 537–544.

[11] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, *Path Planning and Trajectory Planning Algorithms: A General Overview.* Cham: Springer International Publishing, 2015, pp. 3–27. [Online]. Available: https://doi.org/10.1007/978-3-319-14705-5_1

[12] C.-Y. R. Huang, C.-Y. Lai, and K.-T. T. Cheng, "Chapter 4 - fundamentals of algorithms," in *Electronic Design Automation*, L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, Eds. Boston: Morgan Kaufmann, 2009, pp. 173–234. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123743640500114

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[14] R. Matai, S. P. Singh, and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling salesman problem, theory and applications*, vol. 1, no. 1, pp. 1–25, 2010.

[15] E. Aarts, J. Korst, and W. Michiels, *Simulated Annealing.* Boston, MA: Springer US, 2005, pp. 187–210. [Online]. Available: https://doi.org/10.1007/0-387-28356-0_7

[16] G. Gutin and D. Karapetyan, "Generalized traveling salesman problem reduction algorithms," *Algorithmic Operations Research*, vol. 4, no. 2, pp. 144–154, 2009.

[17] S. L. Smith and F. Imeson, "GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem," *Computers & Operations Research*, vol. 87, pp. 1–19, 2017.

[18] J. Schmidt and S. Irnich, "New neighborhoods and an iterated local search algorithm for the generalized traveling salesman problem," *EURO Journal on Computational Optimization*, vol. 10, p. 100029, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2192440622000053

[19] C. E. Noon and J. C. Bean, "An efficient transformation of the generalized traveling salesman problem," *INFOR: Information Systems and Operational Research*, vol. 31, no. 1, pp. 39–44, 1993.

[20] W. Hämäläinen, "Class np, np-complete, and np-hard problems," *Sort*, pp. 1–7, 2006.

[21] B. J. Oommen and L. G. Rueda, "A formal analysis of why heuristic functions work," *Artificial Intelligence*, vol. 164, no. 1, pp. 1–22, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370205000214

[22] D. Levy and M. Newborn, *How computers play chess.* USA: Computer Science Press, Inc., 1991.

[23] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms: generation, enumeration, and search.* Boca Raton, FL: CRC Press, 1999.

[24] E. H. L. Aarts and J. H. M. Korst, "Simulated annealing and boltzmann machines - a stochastic approach to combinatorial optimization and neural computing," in *Wiley-Interscience series in discrete mathematics and optimization*, 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:19877437

[25] F. Glover and M. Laguna, *Tabu search I*, 01 1999, vol. 1.

[26] Z. Shiller, *Off-Line and On-Line Trajectory Planning.* Cham: Springer International Publishing, 2015, pp. 29–62. [Online]. Available: https://doi.org/10.1007/978-3-319-14705-5_2

[27] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[28] A. Schrijver, "On the history of the shortest path problem," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:15086733

[29] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263–, 1998.

[30] S. J. S. J. Russell, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, N.J.: Prentice Hall, 2010.

[31] W. Jiang, "Analysis of iterative deepening a* algorithm," *IOP Conference Series: Earth and Environmental Science*, vol. 693, no. 1, p. 012028, mar 2021. [Online]. Available: https://dx.doi.org/10.1088/1755-1315/693/1/012028

[32] M. Flood and L. Savage, "A game theoretic study of the tactics of area defense," *RAND Research Memorandum*, vol. 51, 1948.

[33] M. Jünger, G. Reinelt, and G. Rinaldi, "Chapter 4 the traveling salesman problem," in *Network Models*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1995, vol. 7, pp. 225–330. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801215

[34] D. L. Applegate, R. E. Bixby, V. Sek, and C. Atal, "Finding cuts in the tsp (a preliminary report)," 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:972108

[35] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM Review*, vol. 33, no. 1, pp. 60–100, 1991. [Online]. Available: https://doi.org/10.1137/1033004

[36] M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming*, vol. 51, no. 1, pp. 141–202, 07 1991. [Online]. Available: https://doi.org/10.1007/BF01586932

[37] D. Rosenkrantz, R. Stearns, and P. II, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.*, vol. 6, pp. 563–581, 09 1977.

[38] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, June 1992. [Online]. Available: https://ideas.repec.org/a/eee/ejores/v59y1992i2p231-247.html

[39] G. Reinelt, "The traveling salesman: computational solutions for tsp applications," 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:120717981

[40] S. Lin, "Computer solutions of the traveling salesman problem," *The Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[41] D. S. Johnson, "Local optimization and the traveling salesman problem," *Lecture Notes in Computer Science*, vol. 442, pp. 446–461, 1990.

[42] M. W. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Operations Research Letters*, vol. 6, pp. 1–7, 1987.

[43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[44] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 06 1953. [Online]. Available: https://doi.org/10.1063/1.1699114

[45] A. Henry-Labordere, "The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem rairo, vol," *The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem RAIRO vol*, 1969.

[46] M. Zia, Z. Cakir, and D. Z. Seker, "Spatial transformation of equality – generalized travelling salesman problem to travelling salesman problem," *ISPRS International Journal of Geo-Information*, vol. 7, no. 3, 2018. [Online]. Available: https://www.mdpi.com/2220-9964/7/3/115

[47] S. S. Srivastava, S. Kumar, R. C. Garg, and P. Sen, "Generalized travelling salesman problem through n sets of nodes," *Can. Oper. Res. Soc.*, vol. 7, pp. 97–101, 1969.

[48] G. Laporte and Y. Nobert, "Generalized travelling salesman problem through n sets of nodes: An integer programming approach," *INFOR: Information Systems and Operational Research*, vol. 21, no. 1, pp. 61–75, 1983.

[49] R. M. Karp, *Reducibility Among Combinatorial Problems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–241. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0_8

[50] J. Saskena, "Mathematical model of scheduling clients through welfare agencies," *Journal of the Canadian Operational Research Society*, vol. 8, pp. 185–200, 1970.

[51] V. Dimitrijević, M. Milosavljević, and M. Marković, "A branch and bound algorithm for solving a generalized traveling salesman problem," *Publikacije Elektrotehničkog fakulteta. Serija Matematika*, pp. 31–35, 1996.

[52] M. Fischetti, J. J. Salazar González, and P. Toth, "A branch-and-cut algorithm for the symmetric generalized traveling salesman problem," *Operations Research*, vol. 45, no. 3, pp. 378–394, 1997. [Online]. Available: https://doi.org/10.1287/opre.45.3.378

[53] Y.-N. Lien, E. Ma, and B. W.-S. Wah, "Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem," *Information Sciences*, vol. 74, no. 1, pp. 177–189, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020025593901337

[54] V. Dimitrijević and Z. Šarić, "An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs," *Information Sciences*, vol. 102, no. 1, pp. 105–110, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025596000849

[55] A. Behzad and M. Modarres, "A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem," in *Proceedings of the 15th International Conference of Systems Engineering*, 2002, pp. 6–8.

[56] M. El Krari, B. Ahiod, and Y. B. El Benani, "A pre-processing reduction method for the generalized travelling salesman problem," *Operational Research*, vol. 21, pp. 2543–2591, 2021.

[57] P. Slavik, "On the approximation of the generalized traveling salesman problem," *Rapport technique, Department of Computer Science, SUNY-Buffalo*, 1997.

[58] N. Garg, G. Konjevod, and R. Ravi, "A polylogarithmic approximation algorithm for the group steiner tree problem," *Journal of Algorithms*, vol. 37, no. 1, pp. 66–84, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196677400910964

[59] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," 1976.

[60] B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol, "Approximation algorithms for generalized mst and tsp in grid clusters," in *Combinatorial Optimization and Applications*, Z. Lu, D. Kim, W. Wu, W. Li, and D.-Z. Du, Eds. Cham: Springer International Publishing, 2015, pp. 110–125.

[61] M. Khachay and K. Neznakhina, "Towards a PTAS for the generalized TSP in grid clusters," *AIP Conference Proceedings*, vol. 1776, no. 1, p. 050003, 10 2016. [Online]. Available: https://doi.org/10.1063/1.4965324

[62] J. Renaud, F. F. Boctor, and G. Laporte, "A fast composite heuristic for the symmetric traveling salesman problem," *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 134–143, 1996.

[63] J. Renaud and F. F. Boctor, "An efficient composite heuristic for the symmetric generalized traveling salesman problem," *European Journal of Operational Research*, vol. 108, no. 3, pp. 571–584, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221797001422

[64] L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221705002523

[65] B. Bontoux, C. Artigues, and D. Feillet, "A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem," *Computers Operations Research*, vol. 37, no. 11, pp. 1844–1852, 2010, metaheuristics for Logistics and Vehicle Routing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054809001488

[66] G. Reinelt, "TSPLIB—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991. [Online]. Available: https://doi.org/10.1287/ijoc.3.4.376

[67] B. Bontoux, C. Artigues, and D. Feillet, "Techniques hybrides de recherche exacte et approchée: application à des problèmes de transport," Ph.D. dissertation, Ph. D. thesis, University of Avignon and the Vaucluse, 2008.

[68] M. Mestria, L. Satoru Ochi, and S. de Lima Martins, "Grasp with path relinking for the symmetric euclidean clustered traveling salesman problem," *Computers Operations Research*, vol. 40, no. 12, pp. 3218–3229, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054812002122

[69] O. Cosma, P. C. Pop, and I. Zelina, "An effective genetic algorithm for solving the clustered shortest-path tree problem," *IEEE Access*, vol. 9, pp. 15 570–15 591, 2021.

[70] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. (2015) National traveling salesman problems. Accessed on 2024-02-20. [Online]. Available: http://www.math.uwaterloo.ca/tsp/world/countries.html

[71] G. Laporte and F. Semet, "Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem," *INFOR*, vol. 37, no. 2, pp. 114–120, 1999.

[72] D. Karapetyan and G. Gutin, "Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 219, no. 2, pp. 234–251, 2012.

[73] S. Ropke and D. Pisinger, “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows,” *Transportation Science*, vol. 40, pp. 455–472, 11 2006.

[74] “Robotnik Summit XL,” https://robotnik.eu/products/mobile-robots/summit-xl-en-2/, accessed on 2024-02-20.

List of Abbreviations

ACO	Ant Colony Optimization
ALNS	Adaptive Large Neighborhood Search
BKS	Best Known Solution
DP	Dynamic Programming
EGTSP	Equality Generalized Traveling Salesman Problem
GA	Genetic Algorithm
GLNS	Large Neighborhood Search for Generalized TSP
GPS	Global Positioning System
GTSP	Generalized Traveling Salesman Problem
ILS	Iterated Local Search
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
LKH	Lin-Kernighan-Helsgaun
LNS	Large Neighborhood Search
MA	Memetic Algorithm
NP	Nondeterministic Polynomial
RGB	Red Green Blue
RRT	Rapidly-exploring Random Tree
ROS	Robot Operating System
SSD	Solid State Drive
TSP	Traveling Salesman Problem