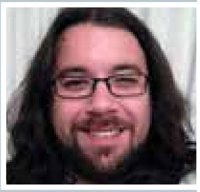
Bargains from travelling salesmen: optimizing a mission

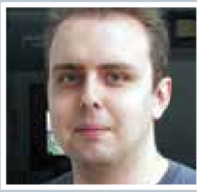
Avi, Kolokolova, Murphy, Bajona, Collingwood, and Reid glide onto the scene.

Tamkin Khan Avi

Antonina Kolokolova

Adam Murphy


Richard Bajona


Kenneth Collingwood

Melissa Reid

Who should read this paper?

Anyone who knows the difficulty of planning an autonomous underwater vehicle (AUV) mission by hand will have a deep interest in this research. As AUV missions grow increasingly numerous and complex, those involved in AUV research – from technicians to scientists to policy-makers – will likely find themselves affected by the automatizing of mission planning. Outside the oceans, anyone naturally inclined toward efficiency and finds suboptimal solutions truly unconscionable will find in this software a kindred spirit.

Why is it important?

AUVs come in various forms. Gliders offer a typical dilemma: what you give up in speed, you gain in range. Electric motored AUVs are typically faster, but gliders use buoyancy, and the trade-off is significant: gliders have very low power consumption and can travel thousands of kilometres, allowing sampling missions of weeks or even months.

Part of the trick is planning a good sampling mission. The authors identify the Travelling Salesman Problem (TSP) as the "heart" of glider mission planning. The problem goes as follows: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? Glider missions, in addition to being an underwater version of the TSP, are plagued by other restrictions: local currents, battery life, and specific time windows for sampling; all conspire to remove degrees of freedom from the mission. The authors' software package employs a variety of solvers to produce a mission plan that optimizes travel time, while also taking into account user-defined additional constraints. The result is an optimal order of goal points, a charted course.

About the authors

Tamkin Khan Avi is completing his M.Sc. in Computer Science at Memorial University of Newfoundland, where he is a Research and Teaching Assistant. This project is part of his M.Sc. research. He was previously a software engineer at KAZ Software. Dr. Antonina Kolokolova obtained her PhD from the University of Toronto, and began teaching at Memorial University in 2007. Her interests include complexity theory and algorithms, in particular scalable approaches to hard problems such as routing and optimization. Adam Murphy is a Computer Science undergraduate student at Memorial University, whose thesis is a result of work on this project. Richard Bajona is a student of Computer Science and Pure Mathematics at Memorial University, where he worked as Research Assistant for Dr. Kolokolova. Kenneth Collingwood is currently completing a B.Sc. in Computer Science at Memorial University, and was involved with the early stages of this project. Melissa Reid completed her B.Sc. in Computer Science at Memorial University. She is currently Director of Cloud Solutions at ID Security Experts Inc., and was involved with the early stages of this project.

GLIDER MISSION PLANNING USING GENERIC SOLVERS

T. Avi, A. Kolokolova, A. Murphy, R. Bajona, K. Collingwood, and M. Reid
Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada

ABSTRACT

In this paper we describe several approaches to the AUV (glider) mission planning problem and investigate their complexity. At the heart of such mission planning are variants of an NP-hard (Non-deterministic Polynominal-time) Asymmetric Travelling Salesman Problem (ATSP); however, some modern-day heuristics can solve this problem optimally in a reasonable amount of time (although providing a proof of optimality slows down the computation). A glider mission plan often has to accommodate a variety of other constraints such as scheduling restrictions, specific time windows or order to visit selected points and so on.

Here we consider a general AUV mission planning problem which, although based on ATSP, can incorporate other constraints. Thus, the use of general-purpose solvers such as Integer Linear Programming or Satisfiability-based solvers may be desired. With this goal, we have developed a software package for the glider mission planning problem that utilizes a variety of existing solvers to compute an optimal order of goal points to visit, subject to travel time as well as user-provided additional constraints. Then, to evaluate feasibility of this setting using state-of-the-art solvers, we analyze the performance of a variety of solvers on the core ATSP problem.

KEYWORDS

AUV mission planning; Gliders; Travelling Salesman Problem; Integer Linear Programming; Pseudo-Boolean Optimization; Satisfiability Modulo Theories

INTRODUCTION

As small autonomous underwater vehicles (AUVs) such as gliders become more accessible and are used for a multitude of tasks, mission planning for such AUVs is becoming more and more challenging. In particular, planning a mission for an AUV that requires visiting a significant number of goal locations can be non-trivial, unless there is a simple ordering on the locations coming from a problem definition. The problem becomes even more complex when planning a mission with multiple AUVs.

As a motivating example, consider a joint mission in which an unmanned aerial vehicle surveys an area and notes a few dozen points of interest, which are then visited by a glider. Suppose, also, that the area happens to have a complex system of currents and land. In that case, planning a mission that would visit these points most efficiently, or determining whether it can be done in a single mission, can become complicated. Indeed, in some locations the currents may be too strong for the glider to fly against them, and other areas may be deemed too unsafe to approach. Planning by hand a mission that could satisfy all such constraints, though it can be done (and is done in practice), may result in a suboptimal solution and becomes unwieldy as the number of points to visit grows.

The goal of our project is automating this mission planning task. However, the underlying problems are NP-hard, and thus no efficient algorithms for them are known. One possibility would be to forfeit optimality and settle for a fast approximation algorithm; instead, we chose to delegate computationally hard tasks to heuristics-based generic solvers. In the last several years such solvers, in particular Satisfiability (SAT) solvers, became a staple method for solving a wide range of constraint satisfaction problems, from automated hardware and software verification to planning problems, to exam scheduling at universities. In this setting, an optimization problem is encoded as an instance of a specific NP-hard problem such as SAT or Integer Linear Programming (ILP), and then approached using heuristics developed specifically for SAT/ILP. So to what extent can solvers' heuristics tackle such complex mission planning problems?

As a part of our project, we have developed a software package to assist with glider mission planning. Users enter the parameters of a glider into the system using a software interface, as well as loading an ocean current map of the desired area. Then waypoints/goal locations can be either uploaded from a file or selected interactively using the interface. After that, a path planner is invoked to compute pairwise distances between goal points. Users then select a desired solver to compute an optimal order of points to visit; an instance of the optimization problem is generated in the format accepted by the solver. After the solver computes the resulting tour (sequence of points), it is displayed in the interface. Once the files encoding the distances and points are generated, users can edit them to add extra constraints, and then pass the result to the solvers. This allows for arbitrary extra constraints (available in that solver's framework) to be incorporated. Please see section entitled "Mission Planner Software" for more details.

Mission planning problem is far from new, and naturally there have been algorithms and software packages tailored to solving it in various contexts. However, many glider missions until recently involved only few (under ten) goals, or goals arranged in a simple pattern such as observation buoys located on a same line. In this project, we are interested in extending this to the case of a large number of goals, and providing a framework that can be extended to multiple (heterogeneous) AUVs and additional constraints.

Based on the goal-based approach from space exploration mission planning, Woodrow et al. [2005] from System Engineering and Assessment Limited (SEA) have developed a goal-based planner for Battlespace Access Unmanned Underwater Vehicle setting. In their paper they discuss an advanced software suite created with the focus on re-planning and goal-based mission planning. Though their software can plan a mission offline, it is generally intended for sophisticated AUVs capable of carrying out computation needed for re-planning on board and operating with a significant degree of autonomy. Whereas we consider a simple mission of visiting a number of locations, their atomic units of planning are of the form "lawnmower search over an area" or "loiter." As a part of their software package, they develop a powerful path planner, capable of multi-parameter optimization (such as risk and energy) in a time-varying field. However, the intended military application naturally limits the availability of their software to the community and it is not clear how it would scale up with the number of goals.

A line of work more appropriate to gliders setting follows Vasilescu et al. [2005] results on using data mule AUVs collecting data from the nodes of an underwater sensor network. Most of this work is concerned with communication protocols and implementation of the framework; however, in a follow-up work by Bhadauria et al. [2011], the data gathering problem is explicitly stated and analyzed algorithmically, albeit not for the underwater sensor networks. In that paper, they describe an approximation algorithm for the data gathering problem based on approximation algorithms for variants of TSP (Travelling Salesman Problem), in particular Euclidean TSP, which does not apply in the underwater setting where currents are present.

An approach similar to ours has been investigated by Drucker et al. [2014] in a context of a different problem: continuous surveillance and monitoring by unmanned aerial vehicles (UAVs). There, given a list of locations to be monitored and maximal allowed times between visits to these locations, as well as flight times and characteristics of the UAVs, the goal is to design a cyclic mission, possibly employing several UAVs, that visits all the locations (repeatedly), satisfying the given constraints. They do discuss TSP as a special case of their problem, where each location needs to be visited only once as opposed to repeatedly. Thus, they do obtain a generalization of TSP with additional constraints. With this formulation of the problem, they experimented with several types of generic solvers, in particular mixed integer linear programming and SAT/SMT (Satisfiability Modulo Theories) solvers that we also employ.

It is worth noting that much of the AUV mission planning literature focuses on the path planning part, as opposed to the ordering of the goal points (see, for example, He et al. [2009]). We have developed simple path planners based on A* and FM* algorithms, but there are much more elaborate path planners for gliders in time-varying fields available, such as Eichhorn [2013]. Our software allows for an external path planner to be used, and we expect users to replace our basic planner with their preferred path planning software with 3D glider motion functionality.

Glider Mission Planning Problem

Consider the following scenario. An ocean survey organization has a multitude of observation buoys. It would like to have an automated way of collecting data from these buoys. It procures a small fleet of gliders, each capable of making contact with a buoy and offloading its data. Now, they would like to schedule data collection missions for their fleet of gliders in an optimal manner. They would like every buoy to be visited, ideally during the course of a single mission. Additionally, they may specify the time limits on visiting certain buoys, time spent downloading data, or have other constraints. Their planning will rely on the information about the currents, weather forecast, parameters of the gliders, and any additional information they can provide to help with the mission planning problem.

There are several settings in which this problem can be viewed; in particular, the planning done on the AUV itself versus the mission planning done offline. Here, we are interested in the offline version, as extra computational power and time available in that case can be used to solve the mission planning problem optimally. More precisely, the formulation of the mission planning problem is as follows.

Given:

1. Location of goal points (in the data collection scenario above, possibly the schedule of the previous data collection times for each buoy, time to receive their data, etc.).
2. The number and physical parameters of the gliders such as speed and battery life (where gliders are not necessarily identical).
3. Starting point(s).
4. A map of the currents, potentially with time-varying information and the weather forecast.
5. Additional constraints.

Compute: An optimal order of the goal points to be visited, if it exists.

Once computed, the points can be uploaded to a glider (with potential intermediate waypoints produced by the path planning software module).

Here, there can be a number of definitions of what constitutes an optimal sequence. The optimization parameters can be the travel time or distance, as well as the number of buoys from which data is collected (possibly weighted by the previous collection times), risk factors (how likely it is for some glider to be lost due to getting caught in a strong current or running out of battery), and other criteria. For the sake of simplicity, here we will optimize the total travel time, as computed by the path planner. In general, we will rely on path planner to provide information about travelling between any two points such as travel time; it can be adapted to provide a combination of risk and travel time weighted by the uncertainty, or other information that can be used to modify the optimization criteria.

This is a classic example of a constraint optimization problem. Here, a number of various constraints are present. The most prominent are linear constraints such as limits on battery life or glider speed. Additionally, there can be Boolean constraints, specifying, for example, that a given buoy must be visited before another, or by a specific AUV. A plethora of other constraints may arise in practice.

Asymmetric TSP Representation

Let us first simplify the problem. Consider a situation where we have a single glider, a single starting point and need to visit all buoys, with no additional constraints; the glider returns back to its start point after completing the mission. Also, suppose the distances (that is, time spent travelling or battery used) between every pair of buoys, as well as to the starting point, are pre-computed and do not change with time. In this case, the problem can be recast as Travelling Salesman (SalesPerson) Problem (TSP). Formally, an input to TSP is a complete graph on n vertices, with all edges labelled with (non-negative) cost values $c(u,v)$. The output in this optimization problem is a minimal "tour"; that is, a sequence involving all vertices in the graph that minimizes the distance travelled.

Although TSP is NP-hard even for the case when the costs on graph edges are 0 or 1, there are dedicated TSP solvers such as Concorde [Cook, n.d.] that perform well in many real-life applications. Also, some additional restrictions such as requiring distances to be Euclidean allow for a very good approximation algorithm [Arora, 1996; Mitchell, 1999]: in our setting, even though the triangle inequality constraint is satisfied, the distances are not metric due to asymmetry, and thus our setting is not Euclidean. Additionally, heuristics such as Lin and Kernighan [1973] and Helsgaun [2000] perform well in practice, although they do not guarantee optimality of the solution. For this project, however, we are interested both in computing an optimal solution and in a possibility to extend a TSP instance with additional constraints. Thus, we will consider more general-purpose solvers.

As time travelled does depend on the direction of travel due to ocean currents, we represent the simplified glider mission planning as an asymmetric TSP problem. Given a map of ocean currents, speed of the glider, and locations of goal points, a path planner computes all pairwise travel times between points. Now, construct a graph with vertices being goal points and the start point, and the cost of each edge $c(u,v)$ a travel time from u to v as computed by the path planner. An optimal tour (ordering) of the vertices of this graph translates into an optimal ordering of the goal points, where the tour is considered to start from the start point.

Note that many of the solvers are designed to work with a symmetric TSP problem. An asymmetric TSP problem is usually approached either using specialized heuristics or by creating an equivalent symmetric TSP instance involving twice as many points [Jonker and Volgenant, 1983].

In general, TSP constraints are not the only constraints a mission plan can require. For example, visiting a goal point is likely to take time, be possibly different for different points, and should be accounted for in the planning. There may be scheduling consideration in the mission plan, requiring certain locations to be visited within a specified time window. TSP constraints themselves can be modified: a very natural modification is TSP with neighbourhoods, where a goal location is represented by a disk with non-trivial radius, and it is sufficient for the AUV to touch this disk at any point.

Thus, a more general framework allowing for encoding of a variety of constraints is needed. There are several widely used choices for such frameworks, each with an associated class of generic solvers.

Integer Linear Programming Formulations
One natural framework for encoding TSP as well as additional constraints is the Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP). In that setting, each constraint is represented as a linear function on the variables to be computed, together with a goal function of these variables to be optimized. Additionally, the variables (or at least some of them in case of MILP) are restricted to be integers. Without that restriction, a solution to a linear program can be found in polynomial time by techniques such as interior point method; the well-known simplex method, although exponential-time in the worst case, performs well in practice. However, ILP itself is an NP-hard problem. Nevertheless, as ILP and MILP problems occur very often in optimization, there is a number of heuristics that can be used to find a solution, although the running time is not guaranteed to be fast in the worst case.

Without additional constraints, (asymmetric) TSP can be encoded as the following polynomial-size integer linear program, by the classic result due to Miller et al. [1960] (known in the literature as "MTZ formulation"). Here, c_{ij} denotes the cost of an edge from v_i to v_j , and a variable x_{ij} is 1 if and only if the edge from v_i to v_j is included in the tour. Variables u_j are supplementary; $u_j = k$ if v_i is k^{th} location visited on the tour.

$$\begin{array}{lll} \text{Minimize} & \Sigma_{i,j} c_{ij} x_{ij} & \text{(MTZ)} \\ \text{Subject to} & \Sigma_{i=1}^{n} x_{ij} = 1, & i \neq j, j = 1, \ldots, n \\ & \Sigma_{j=1}^{n} x_{ij} = 1, & i \neq j, i = 1, \ldots, n \\ & u_i - u_j + (n-1) x_{ij} \leq n-2, & i, j = 2, \ldots, n \\ & 1 \leq u_i \leq n-1, & i = 2, \ldots, n \end{array}$$

Here, the first two groups of constraints state that every site is visited exactly once, and the last group of constraints, the so-called subtour elimination constraints, guarantees that the solution consists of one continuous tour, as opposed to several disjoint cycles.

Subsequently, there have been a number of different formulations of asymmetric TSP as an integer linear program, from variations on the above encoding such as Gouveia and Pires [1999] to formulations via multi-commodity flow; see survey by Oncan et al. [2009] for the list of variants. Of special interest to the AUV community is the time-dependent TSP: there the cost to travel an edge depends on its position in the tour. See Gouveia and Voss [1995] and Miranda-Best et al. [2010] for formulations specific to the time-dependent setting.

Note that MTZ formulation can be easily adapted to the case of multiple AUVs as a multiple TSP problem (mTSP), where a fixed number *m* of AUVs can be used to visit the goals. There, all vertices other than the first still have the constraints $\sum_{i=1}^{n} x_{ij} = 1$ and $\sum_{j=1}^{n} x_{ij} = 1$; however, for the start vertex 1 these constraints become $\sum_{i=1}^{n} x_{i1} = m$ and $\sum_{j=1}^{n} x_{1j} = m$. The subtour elimination constraints and the objective function remain the same. For other formulations of mTSP and multi-depot mTSP (where AUVs can start from different locations) see Bektas survey [2006].

Other extensions of TSP have been considered in practice and encoded in the ILP framework, including variants with time windows and differing costs for different agents.

Satisfiability-Based Approaches

Satisfiability (SAT) problem is one of the most well-studied NP-complete problems, one that was used to encode computation in the result that introduced the very concept of NP-completeness [Cook, 1971]. The classical satisfiability problem is a decision (returning a true/false answer) constraint satisfaction problem. More precisely, the input is a list of constraints of the form "either x_1 or not x_2 or x_3 …," called "clauses." A formula is satisfiable if there is a way to assign values 0,1 to the variables x_i (there exists a truth assignment) so that every constraint contains a variable evaluating to 1 or a negation of a variable evaluating to 0. Checking whether a formula is satisfiable is NP-complete; thus, finding a satisfying assignment is NP-hard. However, there is a plethora of heuristics, implemented by generic SAT solvers that handle practical instances of NP-hard problems such as scheduling and verification surprisingly well.

Powerful as SAT solvers are, there are two issues that they do not address. First, they are tailored towards decision problems rather than optimization. Second, they normally work over Boolean domain rather than integers or real numbers.

Satisfiability Modulo Theories (SMT) is the framework designed to address the second problem. There, a variety of constraints such as arithmetic inequalities are allowed. They are treated as propositional variables from SAT solver point of view, but then the resulting satisfying assignment is checked to see if it makes sense from the point of view of the underlying theory. For example, with Integer Linear Arithmetic as an underlying theory for a formula ($x = 5$ OR NOT $x + 1 > 3$) the SAT solver would consider a formula (p OR NOT q), and find a satisfying assignment with $p = 1$ and $q = 0$. Then, it will pass the resulting system $x = 5, x + 1 \leq 3$ to an underlying theory solver that knows how to solve such systems of equations. In this example, the solver will say that this system has no solutions, prompting the SAT solver to look for a different assignment; in particular, $p = 0, q = 0$ works, as any value of $x \leq 3$ will be $x \neq 5$. So, for example, $x = 2$ would be a solution.

For this project, we used Integer Linear Arithmetic and Uninterpreted Function Theory as underlying theories. Since SMT solvers solve decision problems, we used iterative approach to compute an optimal solution.

Alternatively, there is a class of solvers extending SAT solvers called Pseudo-Boolean Optimization (PBO) solvers [Roussel and Manquinho, 2002], which address both of the concerns above: they can take constraints in the form of a linear function of Boolean variables, and also can compute an optimal solution. During the execution, they output feasible solutions as they compute them, eventually converging on an optimal. We use three PBO solvers, namely clasp, Sat4j and SCIP in our experiments.

MISSION PLANNER SOFTWARE

Our mission planning software prototype, named Searistica, aims to provide an interface for solving the glider mission planning problem. The software consists of a web interface, designed to assist users in choosing the goal locations by interactively selecting them on the ocean current map and visualizing the final answer, and a number of interfaces to the solvers computing the optimal tour. It also includes a basic path planner that computes pairwise travel costs between goal locations; the resulting paths can also be visualized within the interface.

The most computationally intensive part of mission planning is solving the underlying constraint satisfaction problem. To facilitate that, Searistica provides users with a choice to invoke one of the several state-of-the-art solvers including Pseudo-Boolean (PBO) solvers [Le Berre et al., 2010; Gebser et al., 2007; Berthold et al., 2009], Incremental Satisfiability Modulo Theories (SMT) [De Moura and Bjørner, 2008] and 0-1 Integer, Mixed Integer Linear Programming solver (CPLEX) for finding an optimal sequence of goals (tour). In each case, the problem of computing an optimal tour given points selected by users is encoded in a manner that a corresponding solver accepts. At this point, users can edit the generated files to add extra constraints.

Running Searistica

Searistica was developed as an ASP.NET web application on Windows platform, interfacing with a relational database to store the data as well as solvers run on a local machine. In order to use it, one first needs to load the two-dimensional ocean current data for visualization into the database (we have tested it with MS SQL), with four columns corresponding to coordinates *x* and *y*, and ocean current in *u* and *v* directions. Then, the web application's source code is loaded to Visual Studio integrated development environment (IDE); running it there opens the application in a web browser. The solvers to be used need to be installed separately.

More details as well as the software itself will be provided at the Searistica website www.cs.mun.ca/~kol/Searistica. The interface is intended to be fairly self-explanatory, and will be described in the rest of this section.

Framework Architecture

Our software package handles all user interactions through a web application interface. It provides all its operations as web services including project creation, preprocessing ocean data, storing ocean data, generating encodings and passing them to solvers, and visualizing the resulting tour. Thus, changes can be easily accommodated by modifying the web interface. We followed service oriented architecture throughout so that it would be convenient to add any external code base or user defined modules to our system. The framework uses industry standard web service communication method JSON, and XML for data exchange. This framework has been developed as a multi-tier application to separate the presentation, logic and optimization layers.

Ocean Current Data Preprocessing and Goal Location Selection

For our experiments, we used model ocean current data in Drog3D format; we worked with historic NetCDF data as well. Users can upload their own ocean current data to our system; latitudinal and longitudinal components of the current (at a fixed depth) are then extracted for each point in the data. At that time, the glider average speed can also be specified.

In our software prototype, we use a simple format for representing the data to be visualized: a list of tuples (*xcoord*, *ycoord*, *U*, *V*), where *xcoord*, *ycoord* are coordinates of a point, and *U*, *V* are the longitudinal and latitudinal components of the ocean current vector. We use a relational database to store the resulting file. This data is then visualized in the interface (see Figure 1).

Aided by the visual representation of the ocean data, users can select their desired locations for the glider to visit. Users can either select those goal points manually from web interface or can upload their mission goal locations as a file (see Figure 1). Finally, each selected location will become a vertex in the complete graph $G = (V,E)$. The web interface also provides an option to visualize the complete graph with travelling costs generated by the path planning stage.

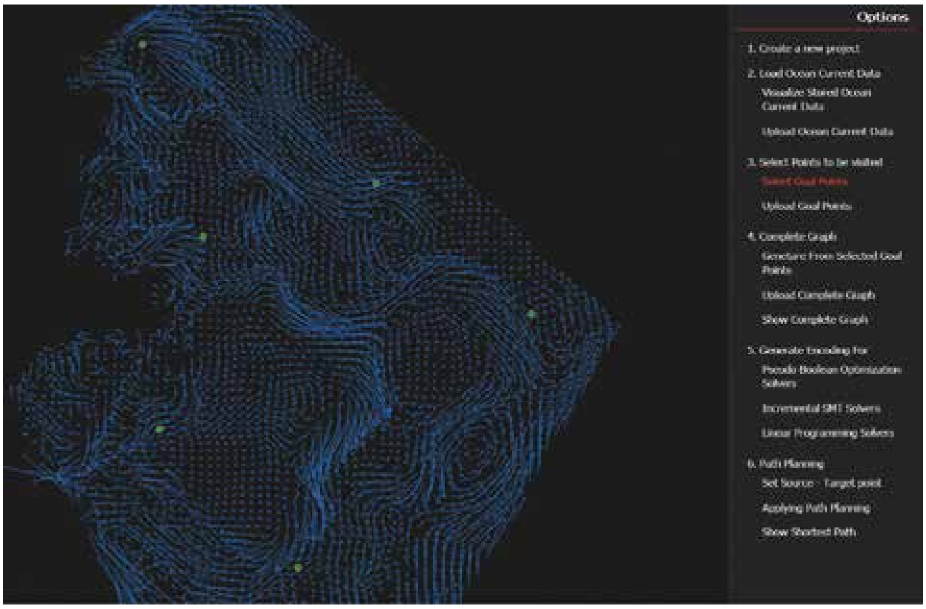

Options
1. Create a new project
2. Load Ocean Current Data
Visualize Stored Ocean Current Data
Upload Ocean Current Data
3. Select Points to be visited
Select Goal Points
Upload Goal Points
4. Complete Graph
Generate From Selected Goal Points
Upload Complete Graph
Show Complete Graph
5. Generate Encoding For
Pseudo Boolean Optimization Solvers
Incremental SMT Solvers
Linear Programming Solvers
6. Path Planning
Set Source - Target point
Applying Path Planning
Show Shortest Path

Figure 1: Ocean data visualization and goal points selection.

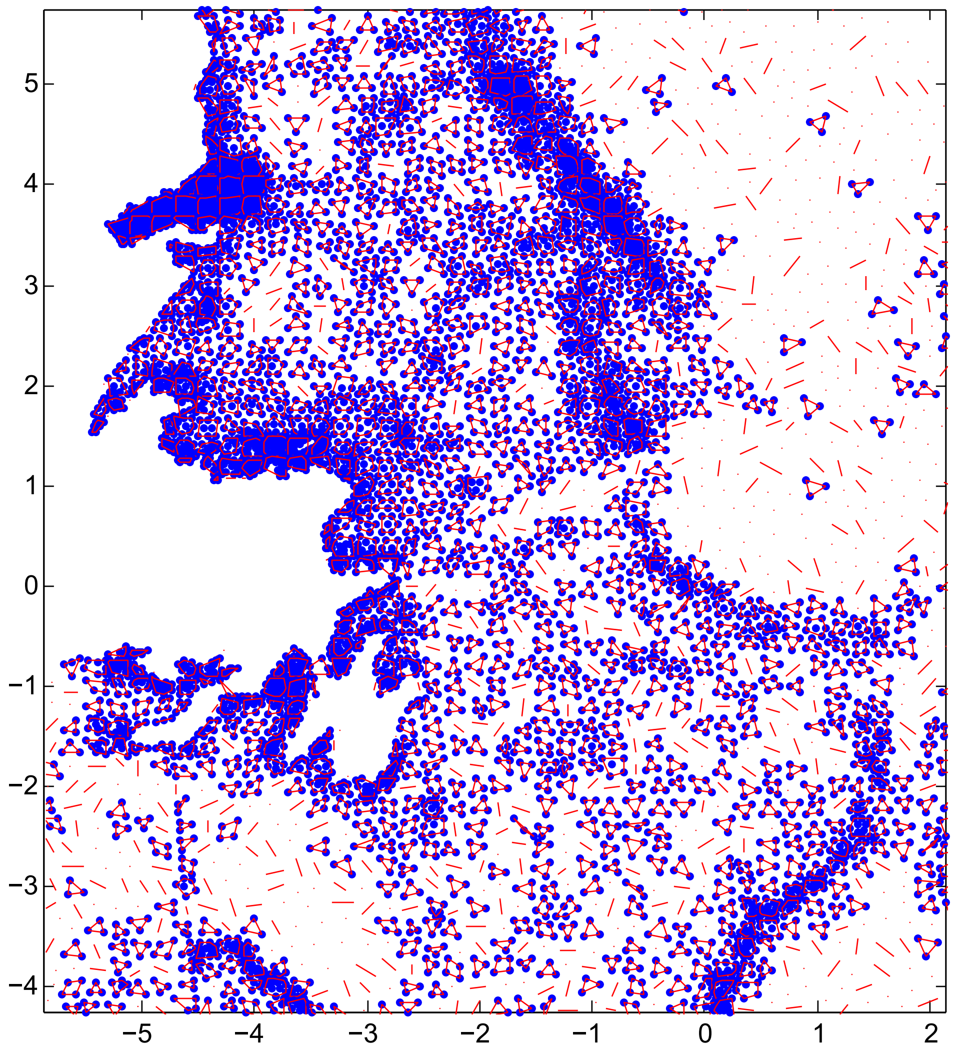


Figure 2: Mapping scattered ocean data to smaller regions.

We used sample data generously provided to us by Dr. Guoqi Han from the Newfoundland Region of Fisheries and Oceans Canada for testing the module, as well as publicly available historic NetCDF data. In both cases, we used scripts to extract the simple format that is loaded into the software for visualization purposes.

Path Planning

Our built-in path planner is grid based; to use it with the data, we average values of currents in each grid cell of user-defined size (set to 1 km in our experiments), and use the centre of the cell as location coordinates. See Figure 2 for the assignment of points in our sample data to the grid, and Figure 3 for the resulting data representation.

Then, our A* algorithm-based path planner is used to compute the costs. To make it faster, we use the A* planner as a path visualization tool, and rely on a recomputed matrix of pairwise cost values for the mission planning. Figure 4 shows the computed path and travelling cost between two nodes v_1 and v_2 in graph. As many users have sophisticated path planners developed in-house, we expect path

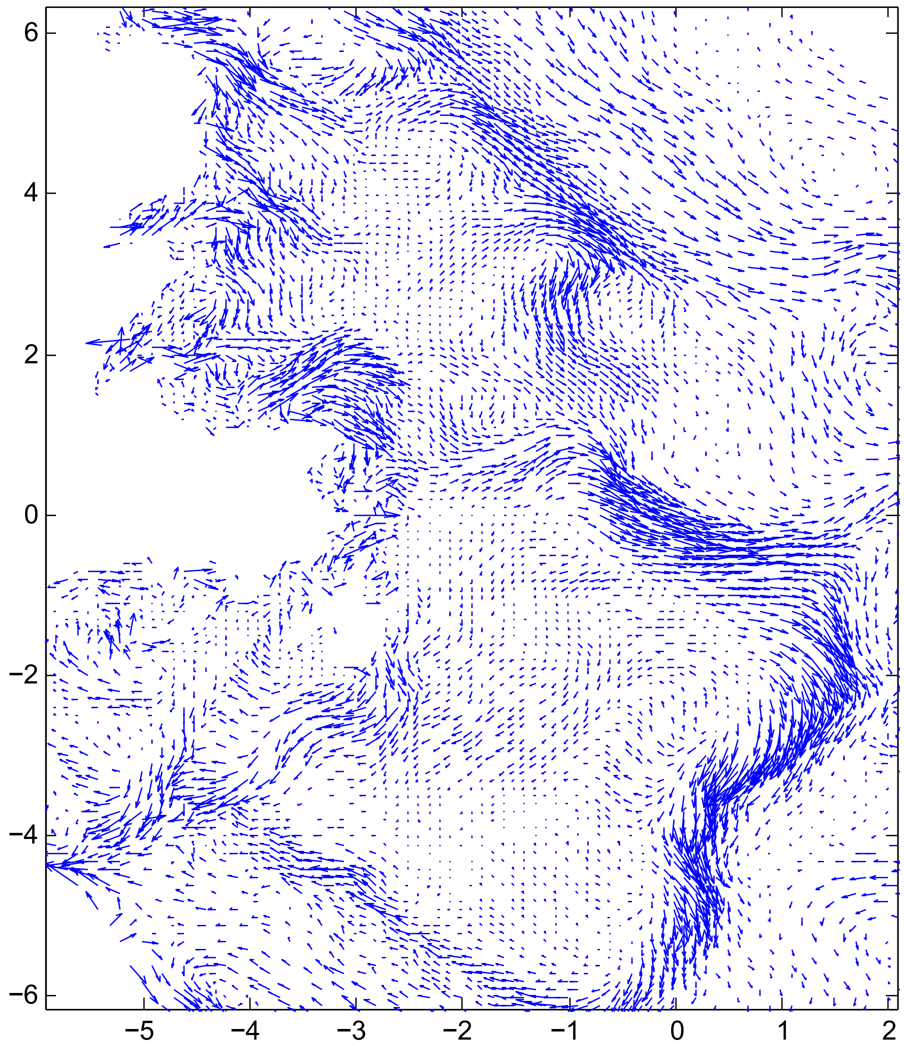


Figure 3: Visualizing average ocean currents on a grid representation.

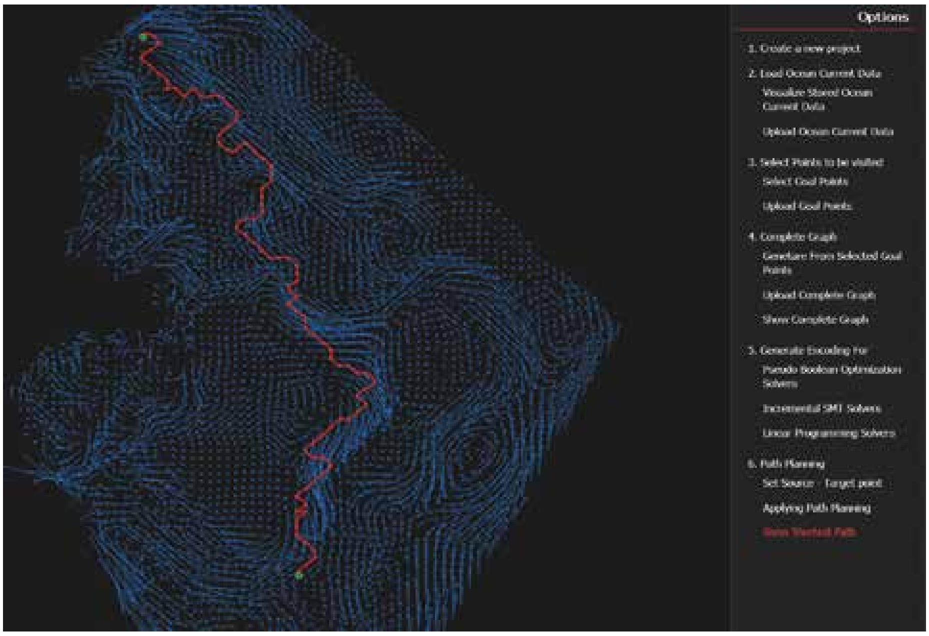
planning task to be passed to an external algorithm, with resulting distance matrix loaded into our software for the optimal tour computation and visualization.

Generating Encodings for the Solvers and Computing an Optimal Tour

The encoding generation and computation of the optimal tour by a solver is the main part of our software package. At this stage, the mission planning problem is encoded using a corresponding formulation such as ILP and the resulting file, after possibly being customized by users, is passed to the respective solver. We have implemented the following types of encodings:

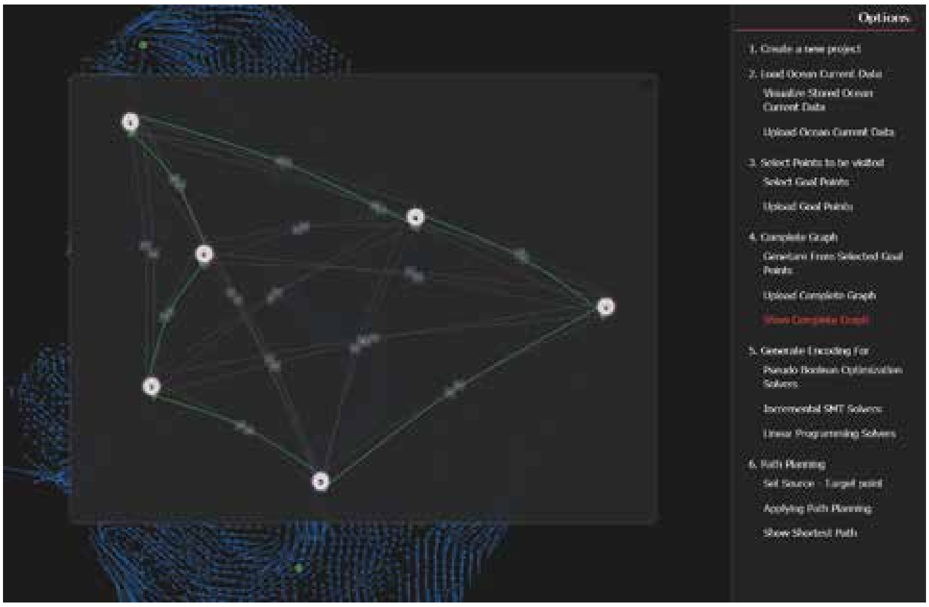
1. Mixed Integer Linear Programming
2. Boolean (0-1) Integer programming and Pseudo-Boolean Optimization (PBO)
3. Satisfiability Modulo Theories (SMT)

When users select the type of the solver, a back-end web server generates the corresponding encoding. At that time, users can do any custom modification to the resulting file such as adding extra constraints.



Options
1. Create a new project
2. Load Ocean Current Data
Visualize Stored Ocean Current Data
Upload Ocean Current Data
3. Select Points to be visited
Select Goal Points
Upload Goal Points
4. Complete Graph
Generate From Selected Goal Points
Upload Complete Graph
Show Complete Graph
5. Generate Encoding For
Pseudo Boolean Optimization Solvers
Incremental SMT Solvers
Linear Programming Solvers
6. Path Planning
Set Source - Target point
Applying Path Planning
Show Shortest Path

Figure 4: Path planner calculates a path of smallest travel cost (time).



Options
1. Create a new project
2. Load Ocean Current Data
Visualize Stored Ocean Current Data
Upload Ocean Current Data
3. Select Points to be visited
Select Goal Points
Upload Goal Points
4. Complete Graph
Generate From Selected Goal Points
Upload Complete Graph
Show Complete Graph
5. Generate Encoding For
Pseudo Boolean Optimization Solvers
Incremental SMT Solvers
Linear Programming Solvers
6. Path Planning
Set Source - Target point
Applying Path Planning
Show Shortest Path

Figure 5: Complete graph generated from selected goal points (Figure 1) with an optimal sequence highlighted.

When a solver returns an optimal solution, the corresponding tour is extracted and visualized using the web interface (see Figure 5).

ENCODING AND INTERFACING WITH GENERIC SOLVERS

The first several encodings are based on the MTZ encoding of TSP described in the equation shown in the section entitled "Integer Linear Programming Formulations." The resulting encoding is then converted into a format accepted by the corresponding solver.

Encodings Based on Integer Linear Programming

Recall that a mixed integer program is an optimization problem in which all constraints are in form of linear equations or inequalities over a subset of real-valued variables ($x_i \in R$), and the objective is to minimize or maximize a linear function $f\ (x_1 ,..., x_n)$. Although solving MILP problems is NP-hard, there is a number of heuristics employed by the solvers used in practice, such as the simplex or branch-and-bound methods.

We encode the TSP problem on the complete graph using the MTZ encoding. That is, the travel cost matrix provides weights of the edges in the objective function, constraints are added for each vertex, and an appropriate number of new variables is introduced together with the subtour elimination constraints.

The simplest encoding is in the *.lp* file format accepted by an MILP solver (for our experiments, we used CPLEX). The resulting text file consist of four parts: “Minimize” followed by the objective function, then “Subject to” followed by the list of constraints, then “Bounds” section providing bounds on variables (in our case, on the variables occurring in the subtour elimination constraints) and finally “Binary” followed by the list of variables that should assume 0-1 values (in our case, these are indicator variables for the edges in the tour). A variable x_k is introduced for each pair (i,j) of the vertices in the graph; that variable will be set to 1 in the resulting solution if and only if the corresponding edge is to be included in the tour. This file can be easily edited by users to add additional constraints.

In the setting of 0-1 Integer Linear Programming, every variable needs to take a value 0 or 1. This is not an issue for the edge indicator variable x_{ij} ; however, variables u_i used in subtour elimination can take integer values up to n. To remedy that, we represent u_i using $log(n) + 1$ Boolean variables, and rewrite the constraints and bounds by substituting u_i with $u_i = \sum_{k=1}^{\log(n)+1} 2^k x_{i,k}$ for new variables x_{ik}. Although this does increase the number of variables, it is only a $log(n)$ increase; however, in this setting some other heuristics become available, in particular heuristics from the realm of Pseudo-Boolean optimization (PBO).

We have used variants of this encoding with CPLEX, as well as several PBO solvers, in particular clasp [Gebser et al., 2007], Sat4j [Le Berre et al., 2010], and SCIP [Berthold et al., 2009]. All these solvers take problem encoding as “.pbo” file format similar to the “.lp” format. See the next section for the performance comparison.

Satisfiability Modulo Theories (SMT) Encoding

We tested both ILA (Integer Linear Arithmetic) [Jovanović and De Moura, 2011] and UF (Uninterpreted Function Theory) encoding on Z3 solver [De Moura and Bjørner, 2008]; we have used SMT-lib2 standard [SMT-LIB, n.d.] for all our SMT encodings.

Integer Linear Arithmetic allows underlying constraints to be represented as linear inequalities. Thus, we again use MTZ formulation of TSP as our encoding; as ILA allows for non-Boolean variables, the u_i’s from MTZ encoding can remain integers. Thus, this encoding is essentially the same as mixed integer linear programming encoding, except rather than optimizing the objective function we add a constraint specifying that it is within a bound, and use the solver to check if such a solution exists. We start with a bound of n * *max of* $\{c_{ij}\}$; at each iteration of the loop, we try for the bound given by the previous solution decremented by 1, as we are searching for the optimal value and our cost values are integers. Note that at each iteration the system retains intermediate information learned at the previous stages.

Uninterpreted Function Theory formulation is not based on MTZ. The encoding we used in this setting considers decision variables v_i corresponding to the order of nodes in the sequence. That is, $v_i = j$ whenever node j is ith node visited. Now, we use a constraint stating that all variables v_i assume distinct values in the range from 1 to n. Finally, we add a constraint $costFunc(v_n, v_1) + \sum_{i=2}^{n} costFunc(v_{i-1}, v_i) \leq bound$, where $costFunc(x_i, x_j) = c_{ij}$ and x_i, x_j both are integer type sort. As this is again a decision problem, we iterate calling the solver with decreased lower bound at each step to find an optimal solution.

SOLVER PERFORMANCE COMPARISON

We have compared the performance of several solvers on instances of the TSP problem generated by our software. Specifically, we used the map of ocean current over Newfoundland and Labrador shelf [Han et al., 2008] with several sets of $n = 4$ to $n = 30$ goal points (randomly generated). In each case, encodings described above were generated and solvers ran with a timeout of 18,000 seconds. All tests were done on Intel Xeon Ubuntu server having 3.50005 GHz of 24 processors with 126 GB of RAM and 890 MB cache.

In Figure 6 the performance of each solver is represented by a different colour plot, describing how many seconds, on average over sets of this size, it took to solve a given instance with n nodes. The lighter green circle in Figure 7 is the average of times when an optimal solution was obtained; a circle is coloured dark green if in at least one experiment the solver found a feasible tour, but did not provide an optimality guarantee before the timeout.

The graph makes it is clear that CPLEX with MILP or even 0-1 encoding performs better than all other solvers; in our experiments, it always provided an optimal solution even for a larger number of points. Surprisingly, CPLEX outperformed other solvers significantly even in the 0-1 integer linear program encoding (see Figure 8 for the comparison of CPLEX (LP) solver on MILP vs. 0-1 ILP encodings). In particular, even on 30 nodes CPLEX could solve the problem within a few seconds.

Pseudo-Boolean solvers except for Wbo performed reasonably well until about 22 nodes, with clasp exhibiting the best performance. Surprisingly SCIP, which implements similar integer linear programming heuristics to CPLEX, did not perform as well. Finally, SMT framework did not result in a viable mission planning, with the likely reason that using iterative approach to find an optimal solution is very inefficient. Moreover, MTZ encoding in the ILA setting performed better than the simpler uninterpreted function theory encoding.

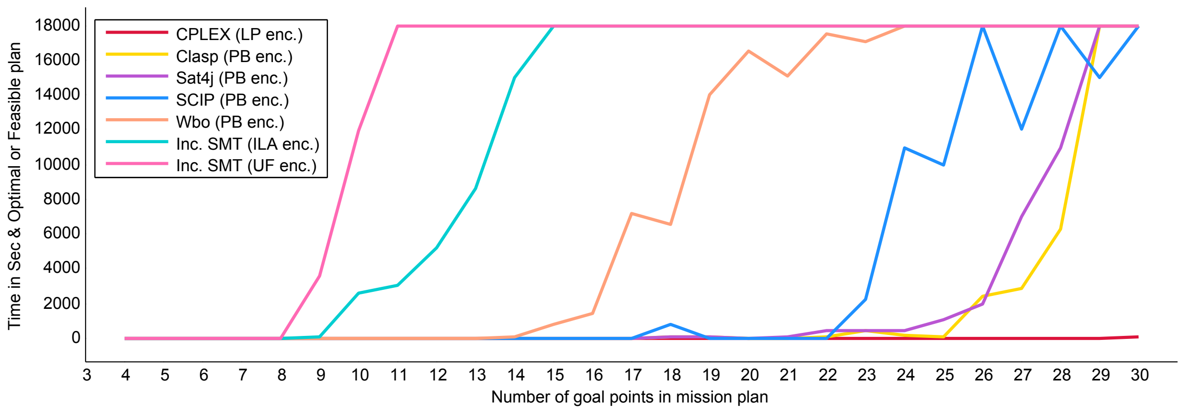


Figure 6: Performance evaluation of different encodings on Linear Programming and Satisfiability based solvers. X-axis represents the number of goal points. Y-axis represents an average time (in seconds) taken by the solver.

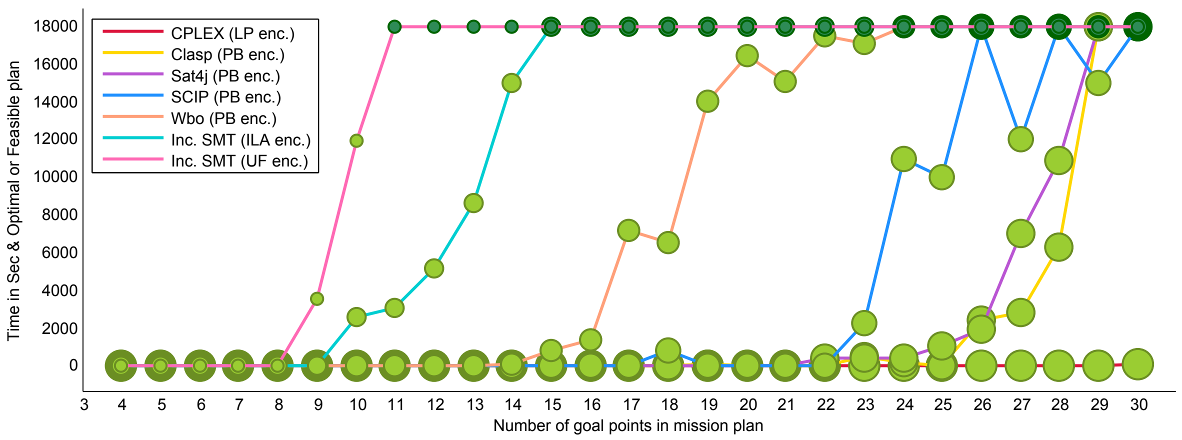


Figure 7: Green circle indicates that the optimal mission plan was found, and dark green that a feasible solution was found without an optimality proof.

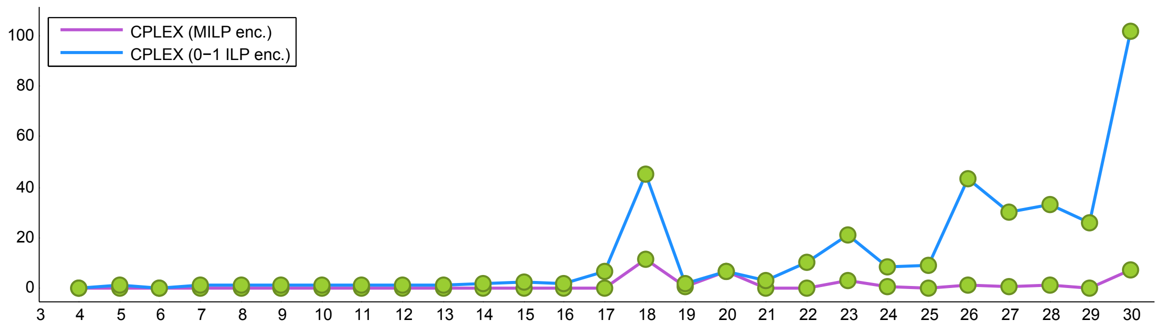


Figure 8: Performance of the CPLEX (LP) solver on MILP and 0-1 ILP encoding of mission plan. X-axis represents the number of goal points in a mission plan.

CONCLUSION

In this paper, we considered the mission planning problem for AUVs in the context of computing an optimal sequence of locations to be visited, with an emphasis on extendibility by a wide variety of other constraints. With that goal in mind, we focused on solving the mission planning problem using generic solvers such as ILP, PBO and SMT solvers. In the course of this project, we built a software prototype package for mission planning.

Our performance comparison indicates that TSP was solvable on the instances we provided; however, only CPLEX solver could provide results within an acceptable timeframe. Thus, unless additional Boolean constraints (encoding, for example, scheduling restrictions) dominate the size of the TSP instance, we suggest using such mixed integer linear programming framework to encode the problem as well as the additional constraints.

Generally, performance of a solver can vary greatly with the type of a formulation of a problem; part of our subsequent work is investigating whether there is a noticeable change in performance with different formulations of TSP, mTSP, TSP with neighbourhoods, and time-dependent TSP. In particular, we are interested in using a time-varying data and analyzing the performance of the solvers on the time-dependent TSP problem.

ACKNOWLEDGMENTS

We would like to thank a number of people for discussions and feedback at various stages of this project, in particular Ralf Bachmayer, Christopher Williams, Moquin He, Michael Eichhorn, Yaroslav Litus, and David Mitchell. We are very grateful to Guoqi Han for giving us access to the sample ocean circulation data. The financial support for this project was provided by the Research and Development Corporation of Newfoundland and Labrador.

REFERENCES

Arora, Sanjeev [1996]. *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*. Foundations of Computer Science. Proceedings of 37th Annual Symposium on IEEE, pp. 2-11.

Bektas, Tolga [2006]. *The multiple Traveling Salesman Problem: an overview of formulations and solution procedures*. Omega, Vol. 34, No. 3, pp. 209-219.

Berthold, Timo; Heinz, Stefan; and Pfetsch, Marc E. [2009]. *Solving Pseudo-Boolean problems with SCIP*. DFG Research Center MATHEON Preprint #600.

Bhadauria, Deepak; Tekdas, Onur; and Isler, Volkan [2011]. *Robotic data mules for collecting data over sparse sensor fields*. Journal of Field Robotics, Vol. 28, No. 3, pp. 388-404.

Cook, William [n.d.]. *Concorde TSP solver*. Retrieved from www.math.uwaterloo.ca/tsp/concorde.html

Cook, S.A. [1971]. *The complexity of theorem-proving procedures*. Proceedings of Third Annual ACM Symposium on Theory of Computing, pp. 151-158.

De Moura, Leonardo and Bjørner, Nikolaj [2008]. Z3: *An efficient SMT solver*. Tools and Algorithms for the Construction and Analysis of Systems. Springer, pp. 337-340.

Drucker, N.; Penn, M.; and Strichman, O. [2014]. *Cyclic routing of unmanned air vehicles*. Information Systems Engineering Technical Reports. IE/IS-2014-02.

Eichhorn, Mike [2013]. *Optimal routing strategies for autonomous underwater vehicles in time varying environment*. Robotics and Autonomous Systems. DOI:10.1016/j.robot.2013.08.010.

Gebser, Martin; Kaufmann, Benjamin; Neumann, André; and Schaub, Torsten [2007]. *clasp: A conflict-driven answer set solver*. Logic Programming and Nonmonotonic Reasoning. Springer, pp. 260-265.

Gouveia, Luis and Pires, Jose Manuel [1999]. *The asymmetric travelling salesman problem and a reformulation of the Miller Tucker Zemlin constraints*. European Journal of Operational Research, Vol. 112, No. 1, pp. 134-146.

Gouveia, Luis and Voss, Stefan [1995]. *A classification of formulations for the (time-dependent) traveling salesman problem*. European Journal of Operational Research, Vol. 83, No. 1, pp. 69-82.

Helsgaun, Keld [2000]. *An effective implementation of the Lin–Kernighan traveling salesman heuristic*. European Journal of Operational Research, Vol. 126, No. 1, pp. 106-130.

Han, Guoqi; Lu, Zhaoshi; Wang, Zeliang; Helbig, James; Chen, Nancy; and De Young, Brad [2008]. *Seasonal variability of the Labrador current and shelf circulation off Newfoundland*. Journal of Geophysical Research: Oceans (1978–2012), Vol. 113, No. C10.

He, M.; Williams, C.D.; and Bachmayer, R. [2009]. *Simulations of an iterative glider mission planning procedure for flying gliders into strong ocean currents*. 16th International Symposium on Unmanned Untethered Submersible Technology.

Jovanović, Dejan and De Moura, Leonardo [2011]. *Cutting to the chase solving Linear Integer Arithmetic*. Automated Deduction–CADE-23. Springer, pp. 338-353.

Jonker, Roy and Volgenant, Ton [1983]. *Transforming asymmetric into symmetric traveling salesman problems*. Operations Research Letters, Vol. 2, No. 4, pp. 161-163.

Le Berre, Daniel and Parrain, Anne [2010]. *The sat4j library, release 2.2, system description*. Journal on Satisfiability, Boolean Modeling and Computation, Vol. 7, pp. 59-64.

Lin, Shen and Kernighan, Brian W. [1973]. *An effective heuristic algorithm for the traveling-salesman problem*. Operations Research, Vol. 21, No. 2, pp. 498-516.

Miranda-Bront, Juan Jose; Mendez-Diaz, Isabel; and Zabala, Paula [2010]. *An integer programming approach for the time-dependent {TSP}*. Electronic Notes in Discrete Mathematics, Vol. 36, No. 0, pp. 351-358.

Mitchell, Joseph S.B. [1999]. *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*. SIAM Journal on Computing, Vol. 28, No. 4, pp. 1298-1309.

Miller, C.E.; Tucker, A.W.; and Zemlin, R.A. [1960]. *Integer programming formulations of traveling salesman problems*. Journal of the Association for Computing Machinery, Vol. 7, No. 4, pp. 326-329.

Oncan, Temel; Altinel, I. Kuban; and Laporte, Gilbert [2009]. *A comparative analysis of several asymmetric traveling salesman* *problem formulations*. Computers and Operations Research, Vol. 36, No. 3, pp. 637-654.

Roussel, Olivier and Manquinho, Vasco [2012]. *Input/output format and solver requirements for the competitions of Pseudo-Boolean solvers*. Retrieved from www.cril.univ-artois.fr/PB12/format.pdf.

SMT-LIB [n.d.]. *The Satisfiability Modulo Theories Library*. Retrieved from http://smtlib.org.

Vasilescu, Iului; Kotay, Keith; Rus, Daniela; Dunbabin, Matthew; and Corke, Peter [2005]. *Data collection, storage, and retrieval with an underwater sensor network*. Proceedings of 3rd International Conference on Embedded Networked Sensor Systems, ACM, pp. 154-165.

Woodrow, I.; Purry, C.; Mawby, A., and Goodwin, J. [2005]. *Autonomous AUV mission planning and replanning – towards true autonomy*. 14th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH.