Glider mission planning using generic solvers

by

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

July 2014

St. John's Newfoundland

Abstract

Autonomous underwater vehicles (AUVs), in particular gliders, have been used for many tasks such as underwater surveys or subsurface sampling. Gliders are small lightweight underwater vehicles, which move by varying their buoyancy. They receive a list of locations to visit at the start of a mission, and then spend most of their time under water, surfacing ocassionally to get their GPS reading and communicate with the control centre, possibly receiving updated instructions.

Modern-day gliders can perform month-long missions. During a mission, several factors impact their performance: ocean currents, lack of communication and GPS while under water and weather. At present, AUV operators do not seem to do much automated mission planning, supplying a glider with a list of goals created by hand. Developing techniques for planning such missions was the goal of this project.

At the heart of such mission planning problem are variants of the Asymmetric Traveling Salesman problem: as it is NP-hard, there is no known algorithm to solve it exactly in polynomial time. However, over the last decade heuristic solvers, in particular solvers based on Integer Linear Programming and Satisfiability, have been used successfully in practice to solve industrial instances of many NP-hard problems.

In this thesis we will describe a glider mission planner that we developed, which is based on using such solvers. We evaluate performance of generic solvers, in particular Mixed Integer Linear Programming (MILP) solver CPLEX, several Pseudo-Boolean Optimization (PBO) solvers Sat4j, Clasp, SCIP and Satisfiability Modulo Theories (SMT) solver Z3. The performance of finding a optimal mission plan also heavily depends on the type of encoding, so in addition to different solvers, we analyzed several types of en-

codings, from the classic Miller, Tucker and Zemlin encoding to Fox, Gavish and Graves' time-dependent TSP encoding. In our experiments, Mixed Integer Linear Programming solver (MILP) has been the most successful compared to other types of generic solvers.

An important question in satisfiability heuristic solver research is modelling the real-world instances by a synthetic distribution. Here, we compared the performance of the solvers on instances of ATSP coming from ocean current data with their performance on crafted instances of random graphs, Euclidean graphs and Euclidean with varying amounts of noise, both symmetric and asymmetric. Developing techniques for planning such missions was the goal of this project, and in this thesis we describe and evaluate several approaches to the AUV mission planning problem.

Acknowledgements

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 (RDC) of NL.

Contents

List of Figures

Chapter 1

Introduction

As small autonomous underwater vehicles (AUVs) such as gliders become more accessible and are used for 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 the problem definition. The problem becomes even more complex when mission planning involve 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 computionally 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.

As a part of our project, we have developed a software package to assist with glider mission planning. A user enters the parameters of a glider into the system using a web 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.

A user then selects 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, the user can edit them adding 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.

A paper based on part of the work presented in this thesis is to appear in the Journal of Ocean Technology, summer 2014 issue. Additional work presented in this thesis includes performance comparisons between different encodings of ATSP (the paper covers only MTZ and SMT Undefined Function Theory encodings), and synthetic data set comparisons.

1.1 Related work

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 10) 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 AUVs and additional constraints.

Based on the goal-based approach from space exploration mission planning, Woodrow, Purry, Mawby and Goodwin from the System Engineering and Assessment Ltd (SEA) have developed a goal-based planner for Battlespace Access Unmanned Underwater Vehicle setting. In their August 2005 paper [WPG05] they discuss an advanced software suite created with the focus on replanning 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 replanning onboard 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 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, Kotay, Rus, Dunbabin and Corke [VKR$^+$05] 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, Tekdas and Isler [BTI11], 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, 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, Penn and Strichman [DPS] 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 an 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 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, Williams and Bachmayer [HWB09]). 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 such as Eichhorn's [Eic13] available. 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.

Chapter 2

Preliminaries

2.1 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 a contact with a buoy and offloading its data. Now, they would like to schedule data collection missions for their fleet of gliders in an 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, 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 inter-

ested 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.
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).

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.

2.2 Representing the problem as Asymmetric TSP

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 precomputed 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 [Coo] which 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 [Aro96, Mit99]: 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-Kernighan [LK73, Hel00] 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 complete 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 [JV83].

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, 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.

2.3 Integer Linear Programming approach

One natural framework for encoding TSP as well as additional constraints is the Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP) [Lue03]. 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, Tucker and Zemlin [MTZ60] (known in the literature as "MTZ formulation"). Subsequently, there has been a number of different formulations of asymmetric TSP as an integer linear program, from variations on MTZ such as Desrochers and Laporte (DL) [DL91], Sherali and Driscoll [SD02] formulation, [GP99] to formulations via multi-commodity flow; see survey by Oncan, Altinel, and Laporte [OAL09] for the list of variants. Of special interest to AUV community is the time-dependent TSP: there, the cost to travel an edge depends on its position in the tour. See [FGG80], [GV95] and [Jua10] 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 $\Sigma_{i=1}^{n} x_{ij} = 1$ and $\Sigma_{j=1}^{n} x_{ij} = 1$; where $x_{ij} = 1$ iff $edge(i, j)$ is on the final tour and otherwise $x_{ij} = 0$. However, for the start vertex 1 these constraints become $\Sigma_{i=1}^{n} x_{i1} = m$ and $\Sigma_{j=1}^{n} x_{1j} = m$. 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 a Bektas'06 survey [Bek06].

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.

2.4 Satisfiability-based approaches

Satisfiability (SAT) problem is one of the most well-studied NP-complete problems, one that was used to encode Turing machines in the result that introduced the very concept of NP-completeness [Coo71]. 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 Undefined 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 solver called Pseudo-Boolean Optimization (PBO) solvers, 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.

Chapter 3

Encodings

ATSP formulations use constraints that ensure that each vertex appears in a tour once, and that the tour edges form a single cycle. Subject to these constraints, the sum of the costs of edges in the tour is minimized. ATSP formulations can be formulated as an assignment problem with cardinality and subtour elimination constraints. Most of these formulations denote by a binary indicator variable $x_{ij} \in (0,1)$ whether an edge E from node v_i to v_j belongs to the final optimal tour. Finally, the objective function $\Sigma_{i,j=1}^{n} x_{ij} c_{ij}$ that has to be minimized, where c_{ij} is travelling cost.

The common polynomial formulation of ASTP is MTZ [MTZ60] encoding described in an equation on page 19. In our experiments we have also used the Desrochers and Laporte (DL) formulation an extended version of MTZ encoding proposed in [DL91] and the Fox, Gavish and Graves (FGG) formulations [FGG80] a time dependent ATSP encoding. Additionally, we devised a non-MTZ-based encoding based on incremental Satisability Modulo Theories (SMT).

In our application, we generate a complete graph in which the weight of an edge between each pair of goal points is the travel cost, computed by running a path planner on the ocean current data.

3.1 Integer Linear Programming encodings

Linear programs form a class of optimization problems (LP) in which the objective function is linear in the unknowns and the constraints consist of linear equalities and inequalities. Any linear program can be transformed into the following compact vector form:

$$\begin{aligned} \text{Minimize} \quad & c^T x \\ \text{Subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Here x is an n-dimensional $(x_1 \ldots x_n)$ column vector, c^T is an n-dimensional row vector, A is an $m \times n$ matrix of co-efficients and b is an m-dimensional column vector. Here $x \geq 0$ means that component of x is nonnegative. The objective is to minimize or maximize a linear function $f = (x_1 \ldots x_n)$. When there are no additional restrictions on the variables $(x_1, \ldots, x_n) \in \mathbb{R}$, then it is an instance of LP, a solution can be found in polynomial time. However, if $(x_1, \ldots, x_n) \in \mathbb{N}$, that is each component is an integer-valued variable, then it becomes an Integer Linear Programming (ILP) problem; if only some x_is are restricted to be integers, then it is Mixed Integer Linear Programming (MILP), and if $(x_1, \ldots, x_n)$ are

further restricted to take only 0/1 values then we obtain the class of problems called 0/1 integer linear programming (0/1 ILP).

Recall that satisfying a CNF formula (SAT solving) is NP-hard. The NP-hardness of 0/1 ILP (and thus also MILP and ILP) can be shown by a reduction from SAT. We can easily convert each CNF clause to corresponding 0-1 Integer Linear Programming constraint. Logical OR can be represent as $(a \vee b) = (a + b) \geq 1$ and negation of literal $(\neg c) = (1 - c)$. For example a CNF clause $(a \vee \neg b \vee c)$ and it's equivalent 0-1 ILP formulation is $(a + (1 - b) + c) \geq 1$.

3.1.1 Constraints shared by different LP encodings

As defined, before the indicator variable $x_{ij} \in \{0, 1\}$ will decide whether an edge from node v_i to v_j will be on the final optimal tour. Here we assume that the tour starts from node 1. We generate $n \times n$ cost matrix, where each element represent the cost c_{ij} of each pairwise goal points generated by heuristic path planner.

$$cost\ matrix = \begin{pmatrix} . & c_{1,2} & c_{1,j-1} & c_{1,n} \\ c_{2,1} & \ddots & c_{2,j-1} & c_{2,n} \\ c_{n-1,1} & c_{n-1,2} & \ddots & c_{n-1,n} \\ c_{n,1} & c_{n,2} & c_{n,3} & . \end{pmatrix}$$

The final objective function that has to be minimized is :

$$\text{Minimize} \qquad \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} c_{ij} x_{ij} \qquad i \neq j \tag{3.1}$$

The in-out degree constraints for each vertex are as follows.

$$\Sigma_{j=1}^{n} x_{ij} = 1, \qquad i \neq j, i = 1, \ldots, n \tag{3.2}$$

$$\Sigma_{i=1}^{n} x_{ij} = 1, \qquad i \neq j, j = 1, \ldots, n \tag{3.3}$$

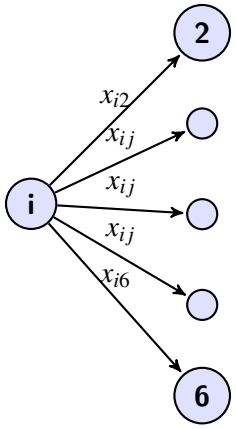

Figure 3.1: Only one of the outgoing indicator variable x_{ij} would be set 1

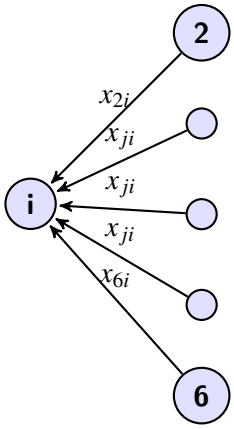

Figure 3.2: Only one of the incoming indicator variable x_{ji} would be set 1

The first equation 3.2 states that each vertex i has exactly one outgoing edge belonging to the tour, and the second equation 3.3 that each vertex i has exactly one incoming tour edge. See figure 3.1 and 3.2

3.1.2 The Miller, Tucker and Zemlin (MTZ) formulation

The constraints described above are not enough, however. The figure 3.3 shows solution satisfying these constraints that consists of several disjoint cycles, which is not desirable if we are interested in a single cycle (tour). Here, by "cycle" in the solution we mean a cycle in a subgraph G_S with only those edges for which $x_{ij} = 1$. The subtour elimination constraints assert that the solution graph G_S consists of a single cycle containing each vertex exactly once, as on figure 3.4.

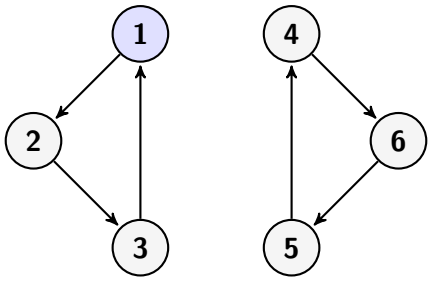

Figure 3.3: The solution graph G_S consists of two cycles: not a valid tour

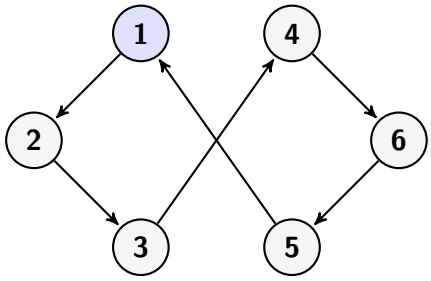

Figure 3.4: The solution graph G_S forms a single Hamiltonian cycle, which corresponds to a valid tour.

The subtour elimination constraints can be defined by introducing new variables $u_2, \ldots, u_n$, encoding the position of a point i in a tour (thus, $2 \leq u_i \leq n \ \ \forall i$). Now, adding constraints

$$u_i - u_j + (n-1)x_{ij} \leq n-2 \qquad \text{(Subtour elimination)}$$

for all pairs $2 \leq i, j \leq n$ guarantees that there is no cycle in the solution not containing first initial starting node v_1. Thus, if this constraint is satisfied then the solution graph G_S contains a unique Hamiltonian cycle.

Now, putting these groups of constraint together, we obtain the classic ILP encoding of ATSP due to Miller, Tucker and Zemlin [MTZ60]:

$$
\begin{array}{lll}
\text{Minimize} & \Sigma_{i,j} c_{ij} x_{ij} & \text{(MTZ)} \\
\text{Subject to} & \Sigma_{i=1}^{n} x_{ij} = 1, & i \neq j, j = 1, \ldots, n \\
& \Sigma_{j=1}^{n} x_{ij} = 1, & i \neq j, i = 1, \ldots, n \\
& u_i - u_j + (n-1) x_{ij} \leq n-2, & i \neq j, (i,j) = 2, \ldots, n \\
& 2 \leq u_i \leq n, & i = 2, \ldots, n \\
& x_{ij} \in \{0,1\} & i, j = 1 \ldots n
\end{array}
$$

Let consider the tour in figure 3.3 having an invalid subtour. Based on the last two subtour elimination constraints of MTZ encoding if we set each tour edge indicator variable $x_{ij} = 1$ then we will get $u_2 - u_3 + 5 * 1 \leq 4$. As $2 \leq u_2, u_3 \leq 6$ thus it is always possible to define u_2 and u_3 such that this constraint is satisfied.

Consider the subtour elimination constraints involving only nodes u_4, u_5 and u_6, with pairs i, j for which $x_{ij} = 1$ (see the second tour in figure 3.3). Then we have these constraints:

$$
\begin{aligned}
(u_4 - u_6 + 5 * 1 \leq 4) \\
(u_6 - u_5 + 5 * 1 \leq 4) \\
+ (u_5 - u_4 + 5 * 1 \leq 4) \\
\hline
(15 \leq 12)
\end{aligned}
\tag{3.4}
$$

Thus, having a subtour not involving intial start node v_1, which will always happen if the solution consists of more than one cycle, gives a contradiction with the subtour elimination constraints.

Now, consider the solution forming a single cycle, corresponding to a valid tour. As the u_i and u_j which are neighbours of intial node v_1 on this cycle (in figure 3.4, u_2 and u_5) appear exactly once in the constraints that have $x_{ij} = 1$, they will not be eliminated by a summation as the previous example. One can show that in this case a feasible solution always exists.

3.1.3 Desrochers and Laporte (DL) formulation

The space of feasible solutions to a linear program forms a polytope. The efficiency of algorithms for finding a solution thus depends on the size of the polytope in an LP relaxation of a given formulation. Desroches and Laporte (DL) [DL91] formulation defines new subtour elimination constraints that produce a more compact polytope than MTZ:

Minimize	$\Sigma_{i,j} c_{ij} x_{ij}$	(DL)
Subject to	$\Sigma_{i=1}^{n} x_{ij} = 1,$	$i \neq j, j = 1, \ldots, n$
	$\Sigma_{j=1}^{n} x_{ij} = 1,$	$i \neq j, i = 1, \ldots, n$
	$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2$	$i, j = 2 \ldots n$
	$1 + (n-3)x_{i1} + \Sigma_{j=2}^{n} x_{ij} \leq u_i \leq n-1-(n-3)x_{1i} - \Sigma_{j=2}^{n} x_{ij}$	$i = 2 \ldots n$
	$x_{ij} \in \{0,1\}$	$i, j = 1 \ldots n$

Desrochers and Laporte ATSP formulation gives a polytope which is a subset of a polytope defined by MTZ encoding. Thus, while the new inequalities provide define the same optimal solution as MTZ, they give a tighter relaxation gap.

3.1.4 Fox, Gavish and Graves (FGG) formulation

Now, consider a scenario where each edge cost depends on the order of that edge on a particular valid tour. The edge cost can be define as c_{ijk} = cost of an edge i, j if it is the k^{th} edge on the tour. This corresponds to the cost of an edge being different depending on the time when it is traversed.

The linear programming formulation we use for this problem is due to Fox, Gavish and Graves [FGG80]. Instead of indicator variables x_{ij} stating whether an edge (i, j) was traversed in the tour, there are index variables r_{ijk} which get the value 1 when the edge (i, j) is traversed as the k^{th} edge on the tour. The rest of the constraints are as follows:

$$
\begin{array}{lll}
\text{Minimize} & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\Sigma_{k=1}^{n} c_{ijk} r_{ijk} & \text{(FGG)} \\
\text{Subject to} & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\Sigma_{k=1}^{n} r_{ijk} = 1, & i \neq j \\
 & \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}\Sigma_{k=1}^{n} r_{ijk} = 1, & i \neq j \\
 & \Sigma_{j=1}^{n}\Sigma_{k=2}^{n} k r_{ijk} - \Sigma_{j}^{n}\Sigma_{k=1}^{n} k r_{jik} = 1 & i = 2 \ldots n \\
 & r_{ijk} \in \{0,1\} & i,j,k = 1 \ldots n
\end{array}
$$

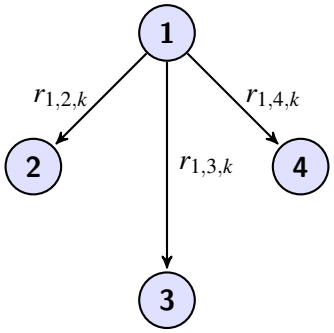

Figure 3.5: All outgoing indicator variables from node 1, only one of these variable will be set to 1

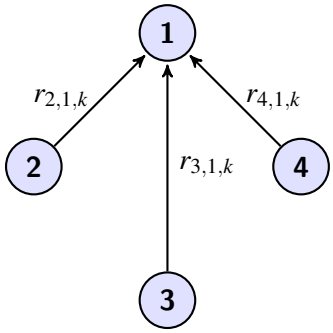

Figure 3.6: All incoming indicator variables at node 1, only one of these variable will be set to 1

The first two constrains in this formulation state that for each vertex i, exactly one of the variables r_{ijk} is set to true. See figure 3.5 and 3.6. The last group of constraints are subtour elimination constraints, forcing the solution to consist of sequential edges forming a valid tour.

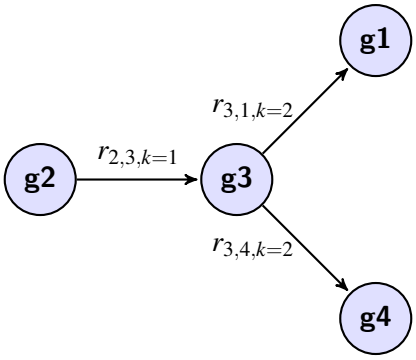

Figure 3.7: The edge from node 2 to node 3 is being visited a $k = 1$ position. The only one outgoing edge from node 3 has to be visited at $k+1 = 2$ position.

The subtour elimination constraints state, for every vertex except for v_1, that the position k in the tour of the incoming and outgoing edge to that vertex differs by 1, with incoming edge position being smaller (see figure 3.7). Now, suppose that there is more than one cycle in the solution graph. Then, consider the first vertex on this cycle (that is, v_i from which an edge with the smallest k in the cycle exits). The incoming edge into v_i will have an index $k' > k$. Thus, the constraint will have non-zero terms $k * r_{i,j,k} - k' * r_{j',i,k'}$. Since $k' > k$, this difference will be < 0, so it cannot be $= 1$. For the graph on figure 3.3, for example, if edge order is $4th : (4,6),\ \ 5th : (6,5),\ \ 6th : (5,4)$, then the vertex 4 will have this problem, with $4 * r_{4,6,4} - 6 * r_{5,4,6} = -2 \neq 1$.

3.2 Encoding integer-valued variables as binary

Pseudo Boolean (PB) Optimization solvers are an extension to core SAT solvers. PBO solves find a model of a conjunction of PB cardinality constraints which optimizes one boolean objective function by converting PB constraints to pure SAT instance. PBO solvers

are based on SAT solving so their approach to find solutions are different that Integer Linear Programming solvers. In the setting of 0-1 Integer Linear Programming, every variable needs to take 0 or 1 value. This is not an issue for the edge indicator variables $x_{i,j}$, 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 $u_i = \Sigma_{k=1}^{log(n)+1} 2^k y_{i,k}$ for new variables $y_{i,k}$. 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). Below encoding is a PBO encoding version of MTZ. Other DL, FGG endings can be converted as same way. Note that when we deal with Binary (0-1) Integer Linear Programming encoding we apply the same approach.

Minimize	$\Sigma_{i,j} c_{ij} x_{ij}$	(PB encoding of MTZ)
Subject to	$\Sigma_{i=1}^{n} x_{ij} = 1,$	$i \neq j, j = 1,\ldots,n$
	$\Sigma_{j=1}^{n} x_{ij} = 1,$	$i \neq j, i = 1,\ldots,n$
	$\Sigma_{k=1}^{log(n)+1} 2^k y_{i,k} - \Sigma_{k=1}^{log(n)+1} 2^k y_{j,k} + (n-1) y_{ij} \leq n-2,$	$i, j = 2,\ldots,n$
	$2 \leq \Sigma_{k=1}^{log(n)+1} 2^k y_{i,k} \leq n,$	$i = 2,\ldots,n$
	$2 \leq \Sigma_{k=1}^{log(n)+1} 2^k y_{j,k} \leq n,$	$j = 2,\ldots,n$

Based on PBO encoding any variable name can be start with x followed by a digit. We have used a mapping table of each readable x_{ij} or y_{ij} indicator variable to corresponding PBO variable xn, where $n \in Int$. We have used variants of this encoding with LP solver

CPLEX, as well as several PBO solvers, in particular clasp [GKNS07], sat4j [LBP+10] and SCIP [BHP09].

3.3 SMT with uninterpreted function theory encoding

A different type of encoding is made possible within SMT with uninterpreted function theory framework. There, the functionality we have available is the ability to directly define a function taking as parameters the values of the variables. Additionally, it is possible to make statements such as ”all values of the following set of variables are distinct”.

However, as SMT is the setting for solving decision problems, we need to specify our problem as ”is there exist an tour with cost less than the given bound?”, and then iteratively find the optimal bound. See section 4.2 for more details on that.

In the following, *Bound* is the bound (initially set to, for example, n times the largest edge cost). The integer-valued variables $p_1 \dots p_n$ denote the positions of vertices in the tour sequence.. That is, $p_j = i$ whenever node j is the i^{th} in the sequence; in particular, we will set $p_1 = 1$ as the first vertex on the tour is assumed to be vertex 1. Finally, we define a function $costFunc(i, j) = c_{ij}$ to give the cost of an edge between v_i and v_j. Note that here we intend to call $costFunc(p_i, p_j)$, with the names of vertices encoded by the variables.

Now, the following constraints encode ATSP in this setting (omitting type declarations)

$$1 \leq p_i \leq n \qquad i = 1, \ldots, n$$

$$Distinct(p_1 \ldots p_n)$$

$$p_1 = 1$$

$$costFunc(i, j) = c_{i,j} \qquad i \neq j, i, j = 1, \ldots, n$$

$$costFunc(p_n, p_1) + \Sigma_{i=1}^{n-1} costFunc(p_i, p_{i+1}) \leq Bound$$

Here, as all values p_i have to be distinct, all vertices will have to appear on the tour. Additionally, again because of distinctness and the order in the last condition, no vertex will repeat (thus avoiding subtours). Note that here the tour is described as a sequence of vertices rather than a sequence of edges as in the previously considered encodings.

Chapter 4

Generic Solvers

Heuristic solvers based on Integer Linear Programming, Satisfiability and Pseudo Boolean Optimisation has become a staple method for dealing with constraint satisfaction [Tsa93], scheduling [GSMT98, GVC95], hardware verification [BLM01], software testing and planning [KS$^+$92, Sur12] problems occurring in practice. Each of these paradigms has its own strengths and weaknesses, depending on the empirical applicability of their heuristics in a specific problem domain. In this chapter, we will discuss the main heuristics used in these classes of solvers.

4.1 Satisfiability (SAT) Solvers

Recall that Boolean satisfiability (SAT) problem is a decision problem where, given an encoding of a Boolean formula, the goal is to decide whether this formula has an assignment of truth values (false/true or 0/1) to its variables that makes the formula evaluate to true.

It is usually assumed, without loss of generality, that the input formula is given in a conjunctive normal form (CNF): that is, it is a conjunction of disjunctions (called "clauses") of literals, where each literal is either a variable or a negation of a variable. For example, in $\phi = (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c)$, the disjuncts $(a \vee b \vee c)$ and $(\neg a \vee \neg b \vee c)$ are two clauses and $\{a, b, c, \neg a, \neg b\}$ are literals.

It is easy to verify that a given truth assignment satisfies the formula (for example, $a = 1, b = 0, c = 1$ satisfies ϕ above), but finding whether there exists such an assignment is much more challenging. SAT is NP-complete, that is, it is a form of a meta-language that can be used to encode any problem with polynomial-time-verifiable solutions. As there is no known algorithm to solve SAT in polynomial time, and no such algorithm is believed to exist, a plethora of heuristics has been developed that seem to perform fairly well on encodings of many practically important instances of problems.

Most SAT solver are based on the basic framework of DP [DP60] and DPLL [DLL62] algorithms. Davis and Putnam first introduced an explicit resolution refutation based iterative variable selection algorithm to find a satisfying assignment for a given CNF formula. Later Davis, Logemann, Loveland proposed a new search based approach over Boolean search tree space. That was more successful and became the basis for most modern SAT solvers. In DPLL algorithm [DLL62], chronological backtracking is performed by applying Boolean Constraint Propagation (BCP). In BCP, if a clause's every literal has been assigned to false except one then force the unassigned variable to true (set 1) to satisfy the clause.

Another successful heuristic that revolutionized SAT solving was CDCL (Conflict Driven clause learning) [BHvMW09, Mar09]. There, the solver learns new clauses from conflicts

during backtrack search [MSS99], and adds them to the formula.

More specifically, CDCL heuristic is applied as follows [Sab12, MSS99]. When a solver finds a conflict (that is, a subformula that is made unsatisfiable by partial assignment so far), it needs to analyze the conflict and learn a new clause to backtrack to height level of decision tree for pruning out large unsatisfiable subtree. CDCL solvers first apply decision making and BCP together, generating a directed acyclic graph called implication graph (IG). There, each vertex represents a variable or literal assignment and an incident edge to a vertex represents the reason (such as a clause) leading to that assignment. In addition to decision making process, CDCL solvers also tag a decision level $d \in (1 \ldots n)$ with each variable values, which means during what level (depth) of the search tree that decision was made. When a conflict happens, resolution refutation in the implication tree is applied until a UIP (Unique Implication Point) has been reached. This point corresponds to a cut in the implication graph. There may be multiple UIP after a conflict and most of the recent SAT solvers only work with first UIP, because it creates a small size of learnt clause.

That process detects the conflict clause, which is then added to the rest of the clauses. After that, the solver jumps back to decision variable and literals assigned at lower decision level. Consider figure 4.3, here $(k \wedge a \wedge \neg f)$ leads to a conflict $c = 0 \wedge c = 1$. So the clause $\neg(k \wedge a \wedge \neg f) = (\neg k \vee \neg a \vee f)$ is added to rest of the clauses. Now it non-chronologically backtracks to decision level 2 $(@d = 2)$ as this is lowest level among other variables decision making levels.

Consider the CNF formula $(x \vee \neg y \vee z) \ \wedge (x \vee y \vee z) \ \wedge (\neg x \vee y \vee z) \ \wedge (a \vee b)$. DPLL algorithm makes the first decision $x = 0$ then makes another decision $z = 0$. Now the clause $(x \vee \neg y \vee z)$ forces the variable $y = 0$ as well as the another clause $(x \vee y \vee z)$ forces

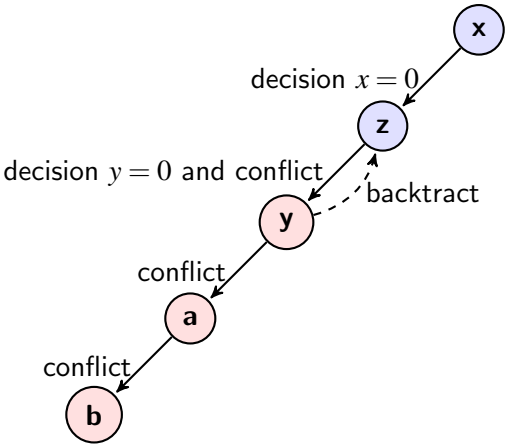

Figure 4.1: DLL: Conflict during decision making

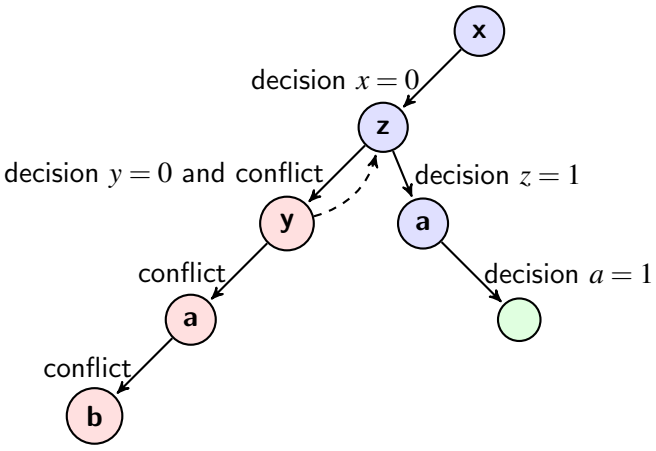

Figure 4.2: DLL: Backtrack and pruning out search tree.

the same variable y must be 1 which is a conflict. Now it backtracks to just immediate previous decision variable z and set it to 1 and finds every clause is true except one. Then assign either a or b true to satisfy the whole CNF formula. See figure 4.1 and 4.2. This approach eliminates the exponential memory requirements problem of DP [DP60] and has seen a lot of success for solving random generated instances.

Most SAT solvers forget clauses exceeding a length threshold at regular intervals to prevent the memory from filling up.

Modern CDCL SAT solvers involve a number of additional techniques such as exploiting structure of conflicts during clause learning [MSS96], periodically restarting backtrack search [GSK98], efficient memory management using watched literals and more. With these heuristics and better memory management techniques with specialized data structures SAT solvers such as MiniSat [ES04] became a staple tool in many industrial applications.

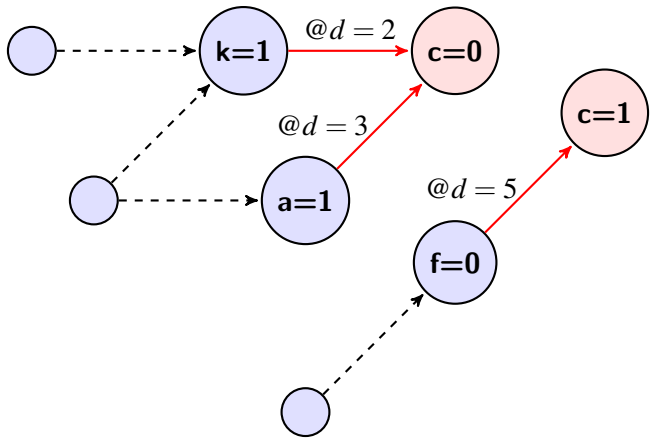

Figure 4.3: Implication Graph

In particular, restarting the search [GSK98, Bie08] also has been useful to solve industrial instances of SAT problem. While restarting the entire search solvers keep all the learnt clauses and again start decision making and BCP. Restarts were earlier used in stochastic local search algorithms and they are necessary for escaping local minima. Different type of special data structure has been proposed for efficient computing during search and one most common lazy data structure is called watched literals [MMZ$^+$01].

4.2 Satisfiability Modulo Theories (SMT) Solvers

SMT is the setting for deciding the satisfiability of a first order logical formula with respect to a different background theory and has wide range of applications in model checking [AMP09], scheduling [BNO$^+$08] etc. In SMT a theory T is defined over a signature Σ, which is a set of function and predicate symbols such as $\{0,1\ldots\cdots+,-,\leq\}$. Predi-

cate symbols in Σ can be used to build theory-atoms $(t-atom)$. A $(t-atom)$ can have theory-terms $(t-term)$ which can be a variables and function symbols in Σ. SMT supports different underlying theory solvers including Equality with Uninterpreted Functions (UF) [NO80, NO07], Integer / Real linear arithmetic (I/R LA), Array (A) [SBDL01], Bit vectors (BV) [BB09] theories. Almost all SMT solvers use SAT heuristics or existing SAT solver as their core decision making process.

There are two different approaches SMT solvers use, called Eager and Lazy approach. The main methodology of the Eager approach is to translate a given SMT problem into equisatisfiable propositional formula and use an off-the-shelf SAT solver as core decision process. It is easy to implement and SAT-solver is used as a black-box, but it potentially generates large encodings and thus slows down the solving. Rather, most of the recent SMT solvers MathSAT, Z3, OpenSMT use lazy approach to solve such problem. In lazy approach each $(t-atom)$ in SMT formula is represented as a pure SAT literal and the resulting formula feed to a SAT engine. When a SAT engine return a satisfying assignment, then underlying corresponding theory solver is used to check for consistency. For example, $(f(a)=b)\wedge(x\vee y)\wedge(m\neq 1\vee m\leq 20)$ can be converted to pure SAT instance by representing each $(t-atom)$ as pure SAT literal $(i\wedge j\wedge k)$ and ask for a model to SAT solver when solver return a model then model is converted back to corresponding theories and check for each theory consistency. This modular approach allows easy different theory combination with smaller encoding generation.

SMT encoding (SMT-LIB [so04]) is in the .smt2 file format accepted by almost every SMT solver. Some other solvers use Simplify format; DIMACS format is also supported [DMB08]. The basic format of these .smt2 files is: at the beginning, $(set-logic\ \ QF_{UFLIA})$

or $(set-logic\ \ QF_{UF})$ etc., has to be defined to tell the solver that what kind of underlying theory would be used to solve given assertions. Then, all the assertion statements are provided. Many of these SMT solvers provide application programming interface (API) to interact with them. We used Z3 [DMB08] solver's python interface to encode our planning problem.

Here we use incremental approach to find the optimal value. We have applied both ILA (Integer Linear Arithmetic) [JDM11] and UF (Uninterpreted Function Theory) encoding on Z3 solver [DMB08]; we have used SMT-lib 2 standard encoding format [so04] for all our SMT encodings. SMT-lib 2 encoding is for promoting a common input and output language for all SMT solvers. Z3 solver [DMB08] compliant with all version of SMT-lib encodings and it provides different high level APIs [z3W].

In SMT-lib encoding integer, real, uninterpreted function all are different type of sorts. Other constraints can be setup with assertion statement. See from below example - the first statement declares an int type sort as variable name $e_{6,5}$, then asserts the constraint $e_{6,5} = 0$ or $e_{6,5} = 1$ as SMT lib encoding and the final assertion represents the equation $u_6 - u_5 + 5 * e_{6,5} \leq 4$. SMT lib representation is useful as these encodings can be parsed easily by solvers in building the expression tree.

$$\begin{aligned}
&\big(declare-const\ \ e_{6,5}\ \ Int\big)\\
&\big(assert((or(=\ \ e_{6,5}\ \ 0)(=\ \ e_{6,5}\ \ 1)\big)\\
&\big(assert(\leq(+(-\ \ u_6\ \ u_5)(*\ \ 5\ \ e_{6,5}))\ \ 4\big)
\end{aligned}$$

In our SMT encoding we setup all initial SMT sorts and transition costs assertions in

different theories and then ask for a valid tour with any cost C. During this first satisfiabilty check solver gains learning lemmas, which will helps it to prune out search trees and chronological backtracking when a new additional assertions are give. When we got initial valid tour of cost C then we assert a new additional constraint - next valid tour constraint should be $< C$, until solver gives UNSAT or timed-out.

4.2.1 Integer linear Arithmetic (ILA) formulation

Linear Arithmetic (LA) theory supports only $+$ and - arithmetical functions over integer or real values. Atoms are of the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n \ \{\leq, <, =\} \ b$ where $(a_1, a_2 \ldots a_n)$ and b are integer or real constant. Decision variables $(x_1, x_2 \ldots x_n)$ are can be either real or integer. LA theory solver supports a different variant of Difference Logic, Mixed Arithmetic. Z3 SMT solver [DMB08] adapts well known Dual Simplex method in general form to solve such arithmetic constraints. It can provide good explanation and support theory propagation during solving instance.

Integer Linear Arithmetic allows underlying constraints to be represented as linear equalities , inequalities in form $(\leq, <)$, and restricts all sorts to be an integer. ILA theory only supports arithmetical functions $+$ and -, but a function could be multiplication of numerical constants and variable sort. Thus, we can use either MTZ, DL or FGG linear programming formulation of TSP as our ILA encoding; as ILA allows for non-Boolean variables, the u_i's from MTZ encoding can remain integer sort. Thus, this encoding is essentially the same as MILP 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.

Consider the conversion of MTZ ILP to incremental ILA. We set each edge variable $e_{i,j}$ as SMT int sort. The second constraint asserts that each variable e_{ij} should be either 0 or 1. Third assertion is the subtour elimination constraints. Final assertion is the tour cost minimization assertion. After that we assert $(check-sat)$ to tell the solver to find a valid tour. At first we just want to find a feasible valid tour with any cost C calculated by SMT solver. Then we (pop) the last minimization assertion and add new minimization assertion $(assert(<= (+(* \ c_{1,2} \ e_{1,2}) \ \ldots\ldots \ (* \ c_{n,n} \ e_{n,n})) \ C-1))$ with a bound by the previous solution decremented by 1. Note that at each iteration the system retains intermediate information learned at the previous stages.

$$
\begin{aligned}
&(declare-const \ e_{i,j} \ Int) && i \neq j, \ for \ i,j = 1,\ldots,n \\
&(assert((or(= \ e_{i,j} \ 0)(= \ e_{i,j} \ 1)) && i \neq j, \ for \ i,j = 1,\ldots,n \\
&(assert(\leq (+(- \ u_i \ u_j)(* \ n-1 \ e_{i,j})) \ n-2) && i \neq j, \ for \ i,j = 2,\ldots,n \\
&(push) \\
&(assert(> (+(* \ c_{1,2} \ e_{1,2}) \ \ldots\ldots \ (* \ c_{n,n} \ e_{n,n})) \ 0)) && i \neq j, \ for \ i,j = 2,\ldots,n
\end{aligned}
$$

4.2.2 Uninterpreted Function Theory (UF) formulation

Uninterpreted Functions (UF) theory uses union-find data structure for checking if a mixture of equalities and disequalities is satisfiable. Given a conjunction of equalities and dis-

equalities between terms using interpreted functions, a congruence closure can be used for representing the smallest set of implied equalities [NO80, NO07]. The UF theory solver state consists of finding a tree structure F that maintains equivalence and disequalities classes. $F(y) = y$ if and only if y is root, otherwise $F(y) = x$, where x is a parent of y. UF theory does two main operations: equivalence relation checking and union operation. In equivalence relation two $(t - trem)$, x and y are equal if $F(x) = F(y)$ means x and y have same root and they are in the same tree. The second operation is union operation - merging the classes of x and y. Consider a simple equalities $f(a, g(y) = f(b, g(z)), a = b, y = z$. Figure 4.4 and 4.5 represent corresponding DAG for two separate terms $f(a, g(y)$ and $f(b, g(z))$ and the equivalent terms $a = b, y = z$ are shown as blue nodes. Uninterpreted terms $g(y)$ and $g(z)$ are congruent because the equivalence relation $y = z$ leads to the same root. The terms $f(a, g(y)$ and $f(b, g(z))$ are also congruent because of another equivalence $a = b$, finally the expression becomes satisfiable. See [dMB09] for more details.

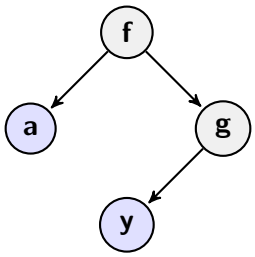

Figure 4.4: DAG For $f(a, g(y)$

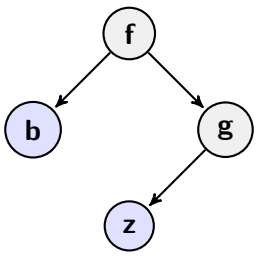

Figure 4.5: DAG For $f(b, g(z))$

Uninterpreted Function Theory is the theory for only equality and dis-equalities uninterpreted function symbols. A function is uninterpreted function if that function is unspecified and has n number of unspecified arguments or arity. For example sort a is an uninterpreted function with 0 arity and $f(x, y)$ is another uninterpreted function with 2 arity x and y sort.

If a set of equalities A and set of disequalities B, then the satisfiability of $A \cup B$ in uninterpreted function theory can be determined by a union-find based abstract operations for manipulating equivalence classes and it has been shown in [DST80, NO80]. Congruence closure algorithms are used to find the satisfying assignments of UF encoding.

Our UF formulation for the glider mission planning problem (or ATSP) is not based on other linear programming formulations. The encoding we used in simple setting considers indicator variables v_i corresponding to the order of nodes in the sequence. That is, $v_i = j$ whenever node j is i^{th} 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 $\Sigma_{i=2}^{n} costFunc(v_{i-1}, v_i) \leq bound$, where $costFunc(i, j) = c_{ij}$.

$$(declare-const\ \ v_i\ \ Int) \qquad i = 1, \ldots, n$$

$$(assert((and\ \ (>=\ \ v_i\ \ 1)\ \ (<=\ \ v_i\ \ n)) \qquad i = 1, \ldots, n$$

$$(assert(\ \ distinct\ \ v_i\ \ \ldots\ \ v_n)) \qquad i = 1, \ldots, n$$

$$(assert(= v_1\ \ 1))$$

$$(declare-fun\ \ costFunc(Int\ \ Int)\ \ Int)$$

$$(assert(= (costFunc\ \ i\ \ j)\ \ c_{i,j})) \qquad i \neq j,\ \ for\ \ i, j = 2, \ldots, n$$

$$(push)$$

$$(assert(> (+(costFunc\ \ v_1\ \ v_2) \ldots (costFunc\ \ v_{n-1}\ \ v_n))\ \ 0))$$

The very first two assertions are defining the sort v_i as Int and specify the bound of

each sort from range $1 \ldots n$. As we know our starting point would be node 1 so we set the assertion $v_1 = 1$. Now we are defining the uninterpreted function $costFunc$ with two Int sort arity and next asserting all traveling costs. Finally providing the minimization statement. As this is again a decision problem, we iterate calling the solver with decreased lower bound at each step by pushing new minimization assertion and popping out the old one.

4.3 Pseudo Boolean Optimization Solvers

Pseudo Boolean Optimization (PBO) solver is an optimisation [BH02] solver over Boolean functions. It extends SAT with cardinality constraints, supporting both linear and non-linear pseudo-Boolean (PB) constraints. Cardinality constraints are in form $\Sigma x_i \geq d$, where d is an integer and each $x_i \in \{0,1\}$ and a pure SAT clause $(a \vee \neg b)$ can be easily represented as linear PB constraint $(a + (1-b)) \geq 1$, where $a, b \in \{0,1\}$. A integer linear pseudo-Boolean constraint can be defined over Boolean variables by $a_1x_1 + a_2x_2 + \cdots + a_nx_n \;\; \geq \;\; b$, where $(a_1, a_2 \ldots a_n)$ and b are integer and decision variables $(x_1 \ldots x_n) \in \{0,1\}$. Pseudo-Boolean problems often contain an objective function, a linear term that should be minimized or maximized under the given constraints. Pseudo-boolean constraints are more expressive and more compact than CNF represent of Boolean formulas.

PBO solvers exploit core SAT solving heuristics and algorithms for finding a satisfying assignment and providing optimization. One way to solve a PB constraint is by transformation to a SAT problem. In [BBR$^+$06] three different techniques, including conversion of constraint into a BDD, network of adders, network of sorters have been proposed for

translating pseudo-Boolean constraints into clauses. Later this approach is used in the PBO solver minisat+.

Another way is to handle PB constraints directly; this approach is taken, in particular, by the solver PBS [ARMS02]. SCIP solver [BHP09] solves PB-problems as a constraint integer programming problem with combining methods from SAT-solving like conflict analysis and restarts. SCIP solver's framework is based on branch-and-bound algorithm to decompose the problem into subproblems, solve a linear relaxation and apply cutting planes method to strengthen the relaxation. Sat4j [LBP$^+$10] PBO solver use core SAT approaches to solve PB constraint problems and is based on Minisat implementation with generic confict driven clause learning engine. Sat4j uses lazy data structure with rapid restarts strategy [Bie08] and conflict clause minimization [SB09]. Finally, Clasp [GKNS07] is an Answer Set Programming solver.

PBO solvers translate a PB constraints to several fragments of decision problems. Finding a optimal solution can be achieved by linear search with a upper bound or binary search with a lower and upper bound. For example optimizing PB objective $\Sigma_{i=1}^{n} c_i x_i$ can be done as follows - the constraint $(c_i x_i + \cdots + c_n x_n > 0)$ is added and converted to CNF and feed to the core SAT solver. If finds a model $M = (x_i \ldots x_n) \in \{0,1\}$ then calculate the objective value $C = \Sigma_{i=1}^{n} c_i x_i$. Then PB solver add a new constraint $\Sigma_{i=1}^{n} c_i x_i < C$ to rest of the PB constraints and again search for a new model M, while keeping all learned constraints. If SAT solver returns UNSAT then the last model is optimum solution. Optimizing by binary search becomes more difficult as finding a lower bound is harder. Sat4j [LBP$^+$10] use linear approach to search for an optimal solution. It has both cutting plane and resolution refutation based PBO solver to deal with PB constraint problems.

PBO solvers accept encoding in the .pbo file format [pbo]. This file format consists of two parts: "min" or "max" then the objective function $\Sigma_{i=1}^{n} c_i x_i$, where each $x_i \in \{0,1\}$, $c_i \in$ *Int* and the rest of the constraints of the form $\Sigma_{i=1}^{n} \{=, \geq, \leq\}\ d$, where d is a integer value. For our experiment we have used Sat4j, SCIP, and Clasp PBO solvers.

4.4 Mixed Integer Linear Programming (MILP) solvers

Mixed Integer Linear programing (MILP) solvers generally use primal-dual simplex method with exact algorithms like branch and bound, cutting planes, interior-point methods for computing linear programming relaxation and heuristic methods such as hill climbing [Lue03]. These solvers do not apply backtracking or constraint learning during search or unit propagation like SAT based solvers. In SAT solving, searching takes place via propagation of the variables domains applying SAT heuristics, but in MILP it takes place by complex but very powerful LP-relaxation, which is computable in polynomial time, although finding an integer solution is NP-hard. CPLEX [Opt93] MILP solver uses both simplex optimizer and the barrier optimizer that exploits a primal-dual logarithmic barrier algorithm for finding the optimal solution.

The encoding is in the .lp file format accepted by an MILP solver (for our experiments, we used CPLEX [IBM, Opt93]). The resulting text file consist of three 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.

Chapter 5

Solver performance comparison

We have compared the performance of several solvers on instances of the TSP problem generated by our software. We randomly generated several sets of $n = 4$ to $n = 30$ goal points. In each case, encodings described above were generated and solvers ran with a timeout of 18000 seconds. All tests were done on Intel Ubuntu workstation with 8 3.6GHz core $i7$ processors and 32Gb of memory. We only perform 8 runs simultaneously, so that the solver runs properly without too much memory swapping or context switching within operating system.

5.1 Ocean data benchmarks

We randomly generates several sets of goal points, then our path planner used the map of ocean current over Newfoundland and Labrador shelf [HLW$^+$08] to calculate pairwise path cost and generates corresponding encoding benchmarks. In figure 5.1 the performance of

each solver on binary MTZ encoding (see section 3.1.2, 3.2) 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 5.2 is the times when an optimal solution was obtained; a circle is coloured dark green if solver found a feasible tour, but did not provide an optimality guarantee before the timeout.

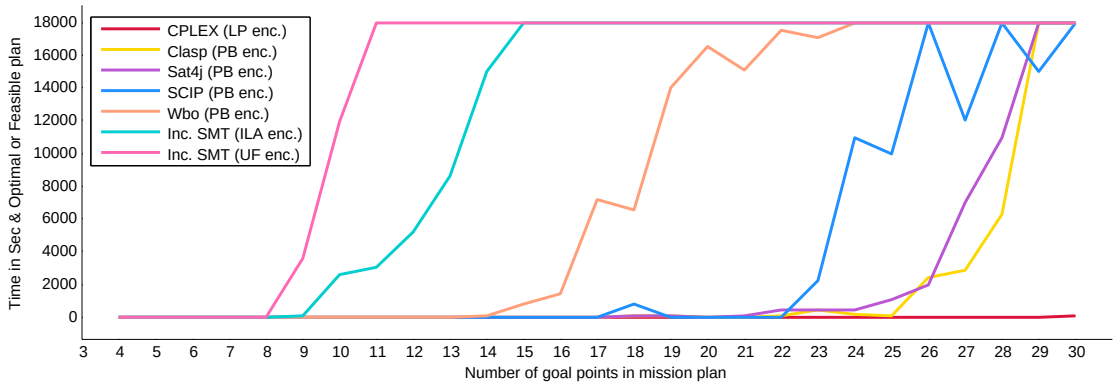

Figure 5.1: Performance evaluation of different encodings. X-axis represents the number of goal points. Y-axis represents an average time (in sec.) taken by solver.

The graph 5.3 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 with both MILP or 0-1 ILP encodings significantly outperformed PBO solvers, even though MILP encoding, as could be expected, resulted in a somewhat better performance. See figure 5.3 for the comparison of CPLEX on MILP vs. 0-1 ILP encodings.

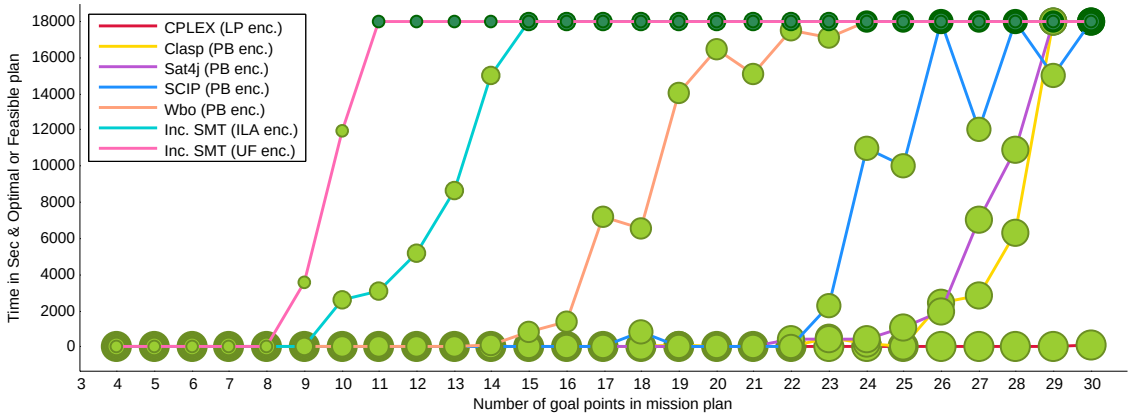

Figure 5.2: Green circle indicates that the optimal mission plan was found, and dark green that a feasible solution was found without an optimality proof.

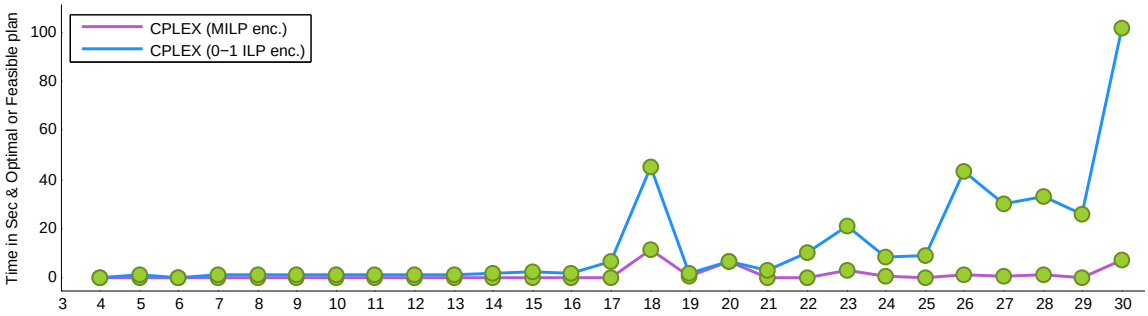

Figure 5.3: Performance of CPLEX solver on MILP and 0-1 ILP encoding of mission plan.

Pseudo-Boolean solvers except for Wbo performed reasonably well until about 22 nodes, with Clasp exhibiting the best performance. 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.

We also generated benchmarks based on DL encoding [DL91] - an extended formulation of MTZ encoding [MTZ60], described in section 3.1.3. Pseudo boolean solver Sat4j was still slower compared to linear programing CPLEX solver, but took less time compared to MTZ encoding, see in figure 5.1 and 5.4. This seems to suggest that reducing the size of the polytope helps even for solvers that do not use simplex method to deal with linear constraints, though on the DL encoding SCIP solver was little better than sat4j solver.

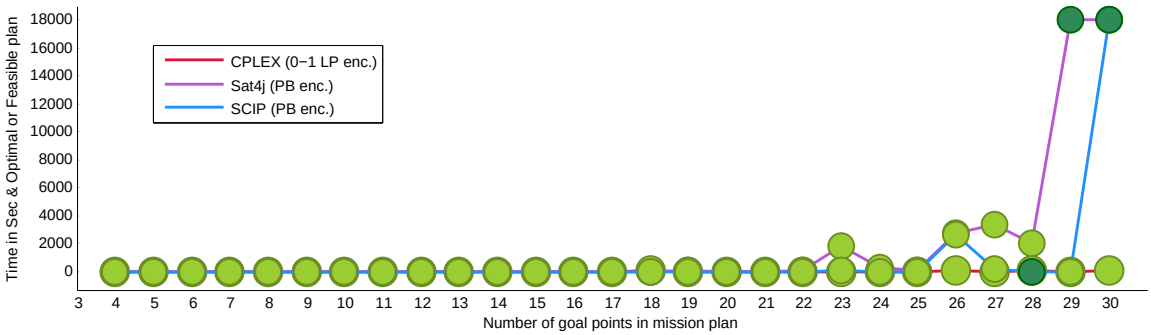

Figure 5.4: DL encoding

For time varying TSP problem benchmark we use FGG encoding [FGG80], described in section 3.1.4. Here the CPLEX solver did the best among other pseudo boolean solvers. Sat4j always gives satisfiable solution on benchmark having more than 15 nodes also clearly lagging behind from other SCIP solver, see figure 5.5.

5.2 Synthetic benchmarks

Our next goal was to see what properties of the ocean data might make it hard or easy for the solvers, and so we generated several synthetic data sets emphasizing various possible

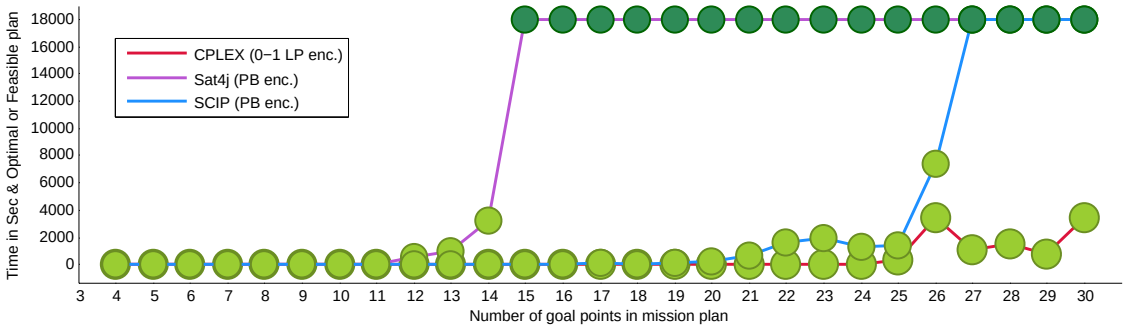

Figure 5.5: FGG encoding

properties of the ocean data, to see how the performance on these data sets compares with the performance on the ocean data. In particular, we created the following data sets:

1. Random data: weight c_{ij} of each edge of a graph is a random number in the range from 1 to 100.

2. Euclidean data: the sets of points on which our ocean data set, except with Euclidean distance rather than distance computed by the path planner.

3. Symmetric random data: generated from he above Euclidean data set points, where weight c_{ij} of each edge of a graph is a random number in the range from 1 to $maxOf(EuclideanCosts)$, and $c_{ij} = c_{ji}$ for all i, j.

4. Noisy Euclidean data: in AsymmetricNoisy data set, half of Euclidean edges cost c_{ij} have been altered asymmetrically by 30% of c_{ij} (that is, $c'_{ij} = c_{ij} + r * c_{ij}$, where $r \in (-0.3, \ldots, 0.3) * c_{ij}$.

5. Noisier Euclidean data: in AsymmetricRandomNoisy data set, half of Euclidean edges cost have been replaced by a random number in the range 1 to $maxOf(EuclideanCosts)$.

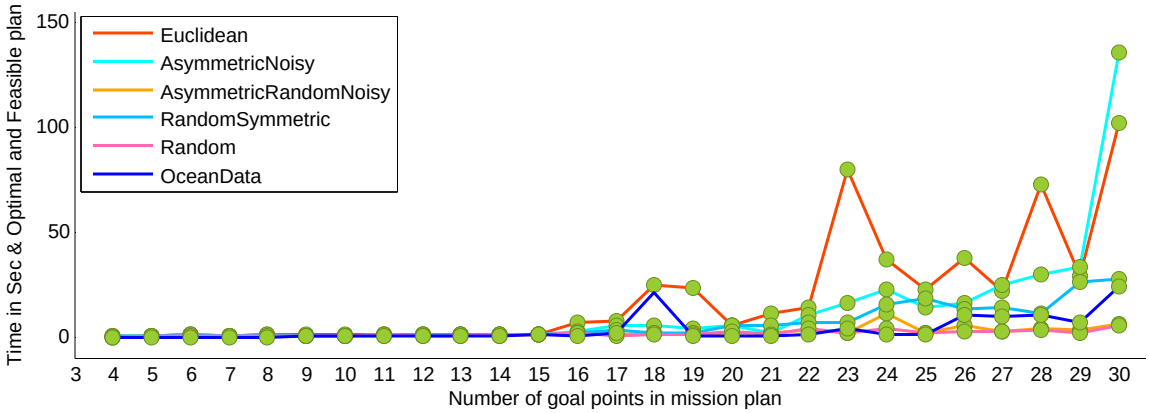

Figure 5.6: Cplex on synthetic benchmarks

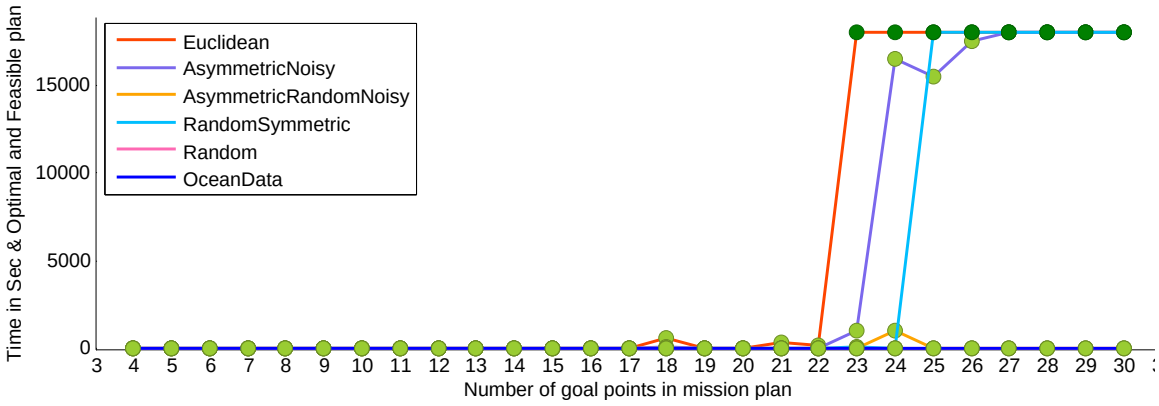

Figure 5.7: SCIP on synthetic benchmarks

For each of the data sets, we performed 5 experiments for each number n of the vertices in the graph, up to 30 nodes. We used MTZ encoding for all our experiments, with 0-1 ILP encoding for CPLEX. Figures 5.6, 5.7 5.9, 5.10 show the best running time of the respective solvers in each of the sets of experiments. In all cases, CPLEX was able to solve the instances much faster than PBO solvers.

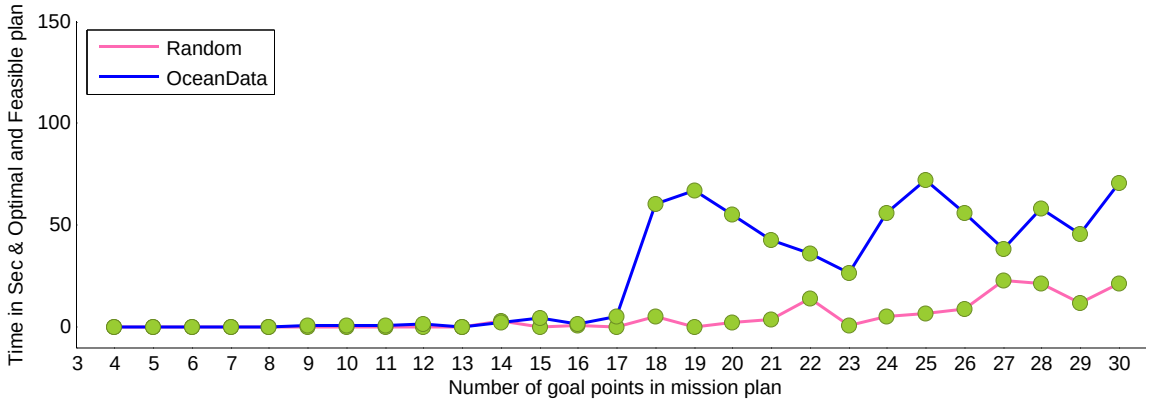

Figure 5.8: SCIP on random and ocean data benchmarks

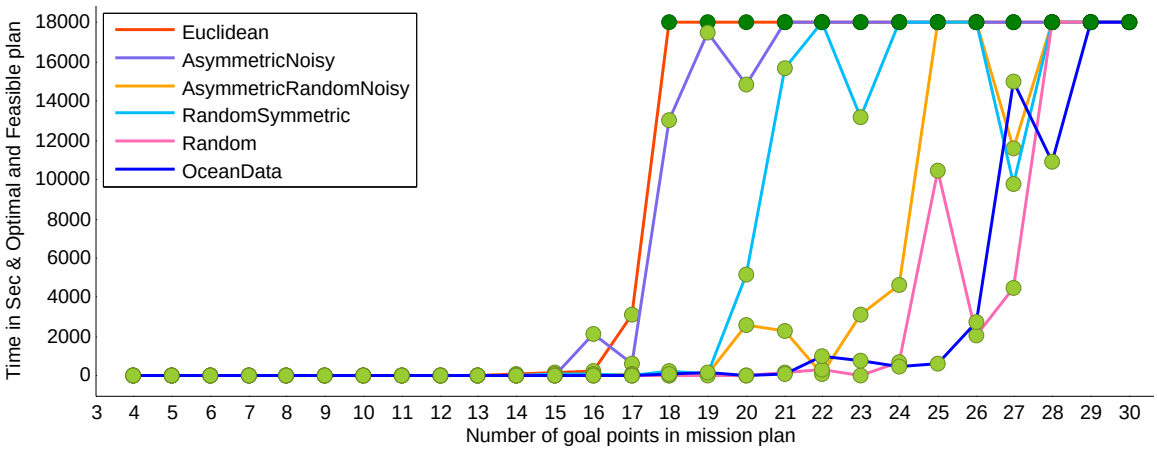

Figure 5.9: Sat4j on synthetic benchmarks

These figures suggest that CPLEX solver finds random data sets to be the easiest, and ocean data set to be relatively easy compared to Euclidean, noisy Euclidean and symmetric random data sets, whereas for PBO solvers Sat4j and clasp, symmetry in the data presented a problem, noticeably increasing the time. In particular, all solvers had their worst performance when the tour was coming from a Euclidean graph, which runs contrary to the fact that TSP on Euclidean graphs is relatively easy, as there are very good approximation

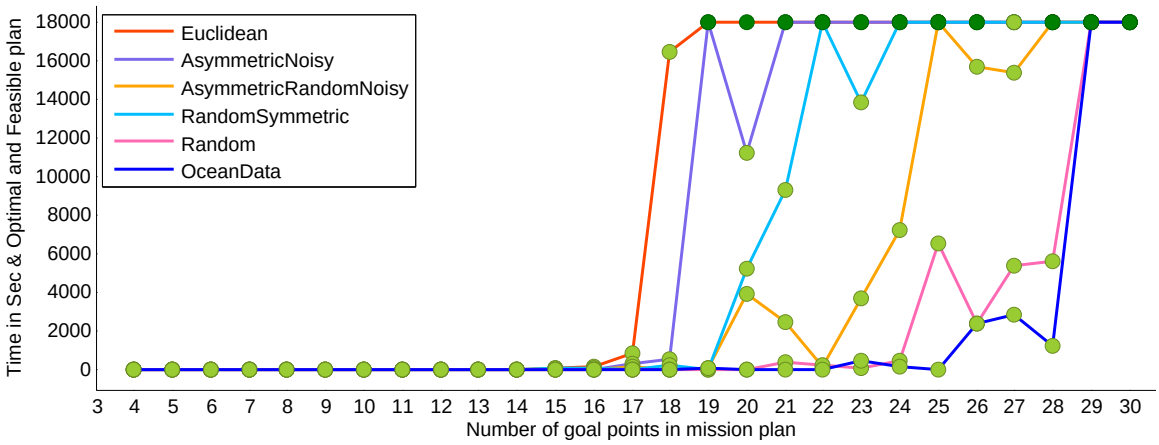

Figure 5.10: Clasp on synthetic benchmarks

schemes for this setting. However, from the point of view of SAT heuristics, it is possibly not too surprising that the symmetries in the instance make it harder.

Overall, performance of CPLEX on the ocean data set seemed to be the closest to AsymmetricRandomNoisy and Random graphs, whereas performance of PBO solvers was most akin to that on random data. More experiments would be needed, however, to verify this intuition and draw statistically significant conclusions.

Chapter 6

'Searistica' Software Implementation

Our mission planning software prototype, named Searistica, aims to provide an interface for solving the glider mission planning problem, given parameters of a glider and a map of ocean currents in the desired area. The goal points can be selected from the interface or uploaded. The software consists of a web interface, designed to assist the user in choosing the goal locations by interactively selecting them on the map visualizing the currents (see figure 6.2) and visualize the final answer, and a number of interfaces to the solvers computing the optimal tour. It also includes a 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, which is TSP in the most basic setting. To facilitate that, Searistica provides user with a choice to invoke one of the several state-of-the-art solvers including Pseudo-Boolean (PBO) solvers [LBP$^+$10, GKNS07, BHP09], Incremental Satisfiability Modulo Theories (SMT) [DMB08] and 0-1 Binary Integer, Mixed Integer

Programming (MIP) solvers for finding an optimal sequence of goals (tour). In each case, the problem of computing an optimal tour of given points selected by the user is encoded in a manner that a corresponding solver accepts. At this point, the user can edit the generated files to add extra constraints.

6.1 How to use Searistica

Searistica was developed as an ASP.NET web application on Windows platform, interfacing with a database to store the data as well as solvers run on a local server. In order to use it, one first needs to load the two-dimensional ocean current data for visualization into the database (we have tested with MS SQL server), with four columns corresponding to coordinates x and y, and current strength in u and v directions. Then, the web application is loaded using a corresponding software such as Visual Studio and displayed in a web browser such as Chrome at the URL http://localhost/AUVMissionPlanning/Default.aspx. 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 http://www.cs.mun.ca/∼kol/searistica.html The interface is intended to be fairly self-explanatory, and will be described in the rest of this section.

6.2 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

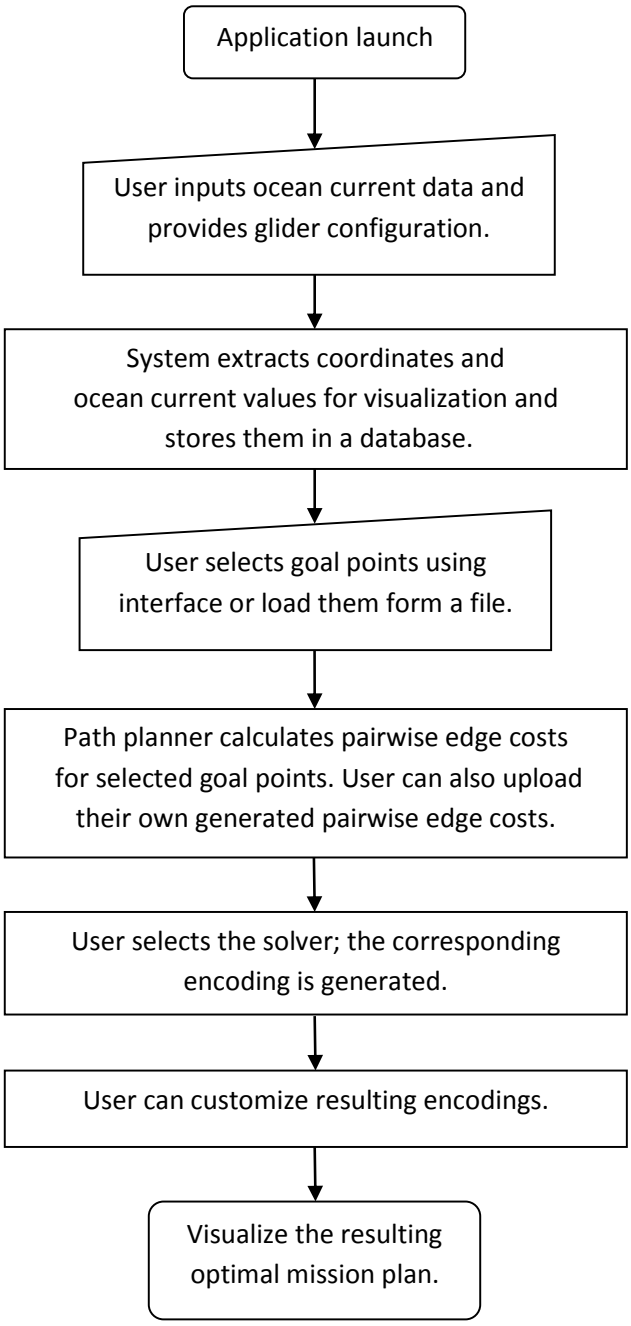

Figure 6.1: Work flow of software framework

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 (SOA) throughout, so that it would be convenient to include 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 was developed as a multi-tier application to separate the presentation, logic and optimization layers. Figure 6.1 outlines the overall work flow diagram.

6.3 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. The user 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 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 6.2).

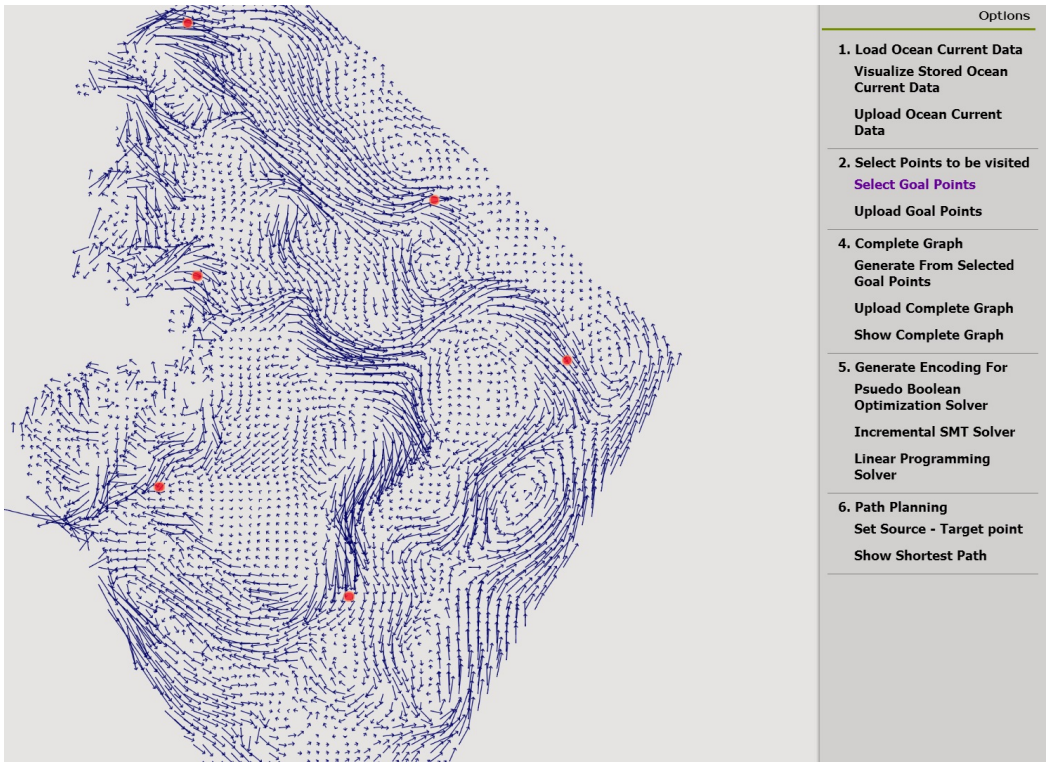

Figure 6.2: Ocean data visualization and goal points selection

Aided by the visual representation of the ocean data, the user can select their desired locations for the glider to visit. Either the user can select those goal points manually from web interface or can upload their mission goal locations as a file (see figure 6.2). 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; see figure 6.6.

We used sample data generously provided to us by Dr. Han from the Newfoundland Department of Fisheries and Oceans 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.

6.4 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 1km in our experiments), and use the centre of the cell as a location coordinates. See figure 6.3 for the assignment of points in our sample data to the grid, and figure 6.4 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 precomputed matrix of pairwise cost values for the optimization. Figure 6.5 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 that they will prefer to load their precomputed pairwise distances matrix for the tour calculation.

6.5 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 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 the user,

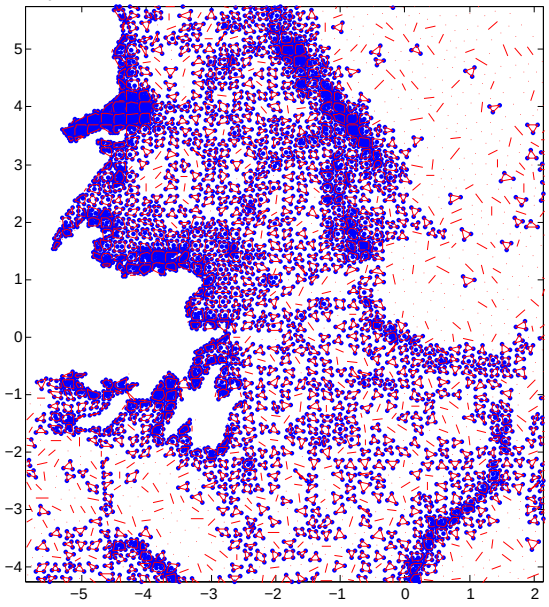

Figure 6.3: Pre-processor maps scattered ocean data to a grid. Each group of connected points corresponds to one grid cell; in the sparser areas, a grid cell can contain one or, rarely, zero points.

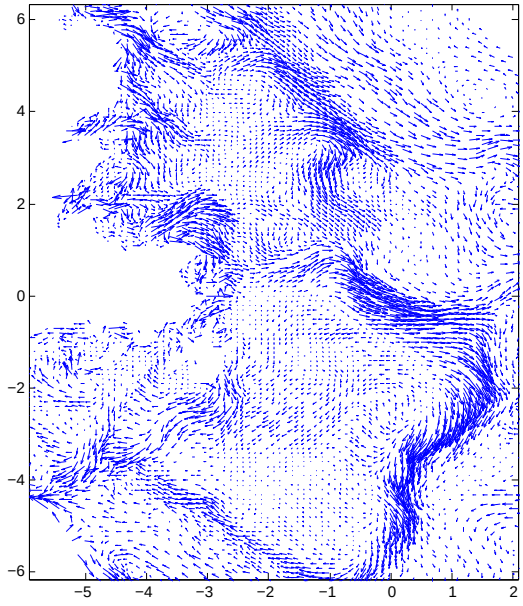

Figure 6.4: After processing, visualizing average ocean currents on a grid.

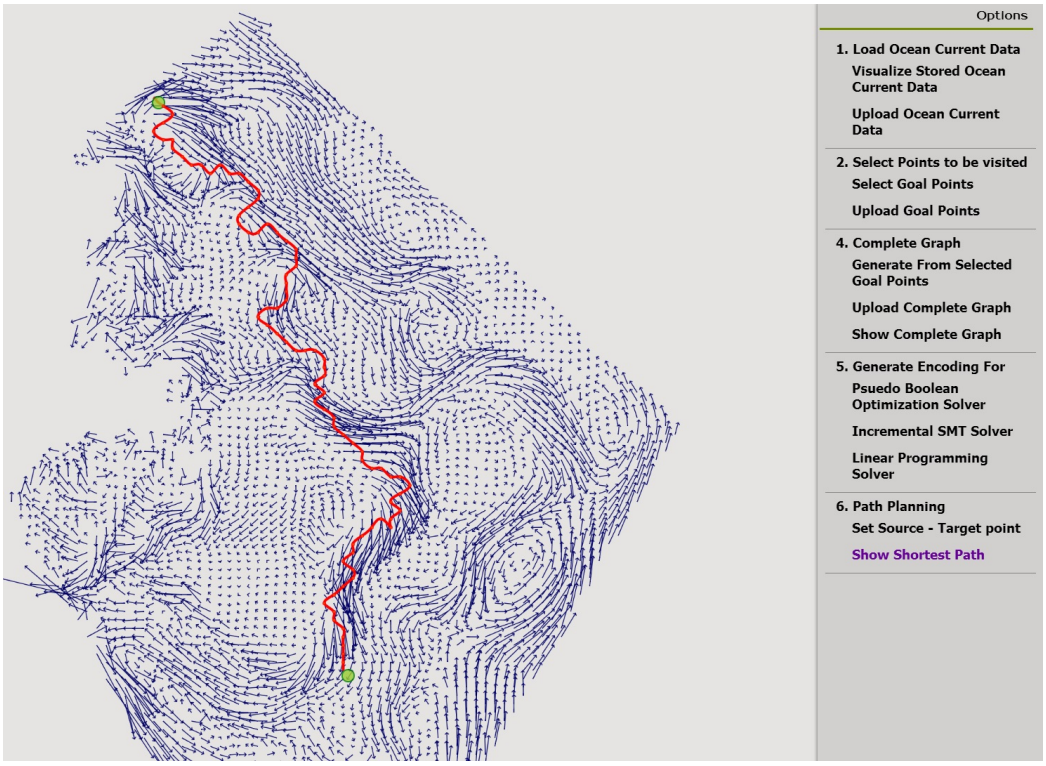

Figure 6.5: Path planner calculates a path of smallest travel cost (time)

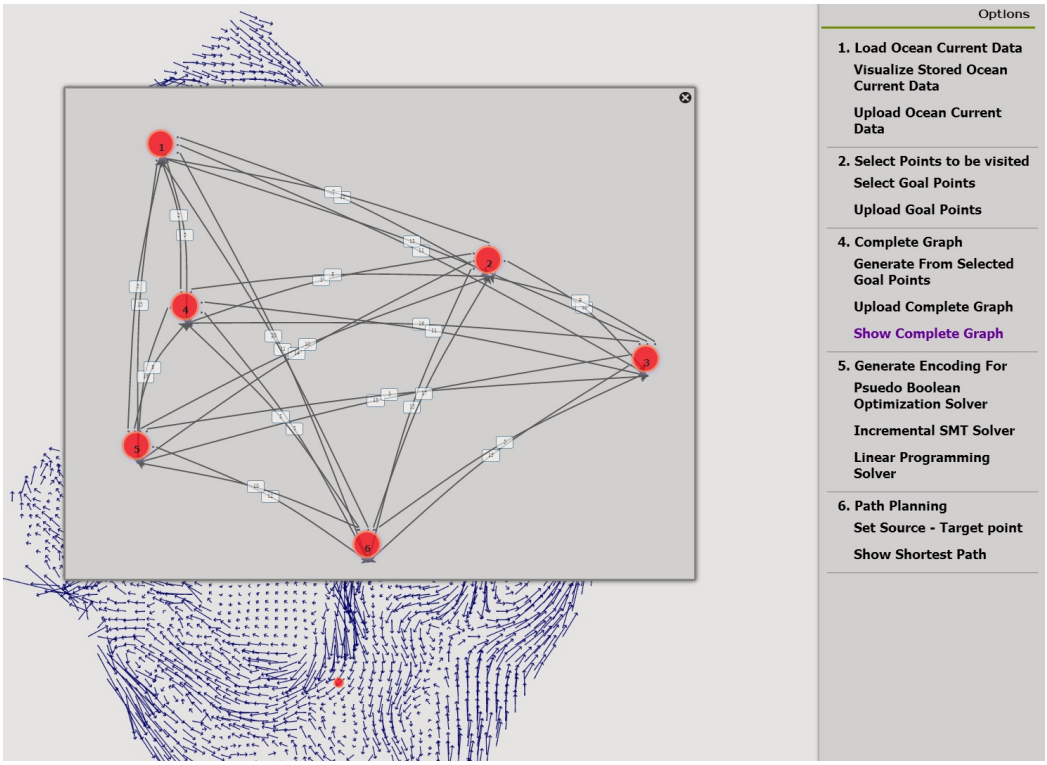

Figure 6.6: Complete graph generated by path planner from selected goal points

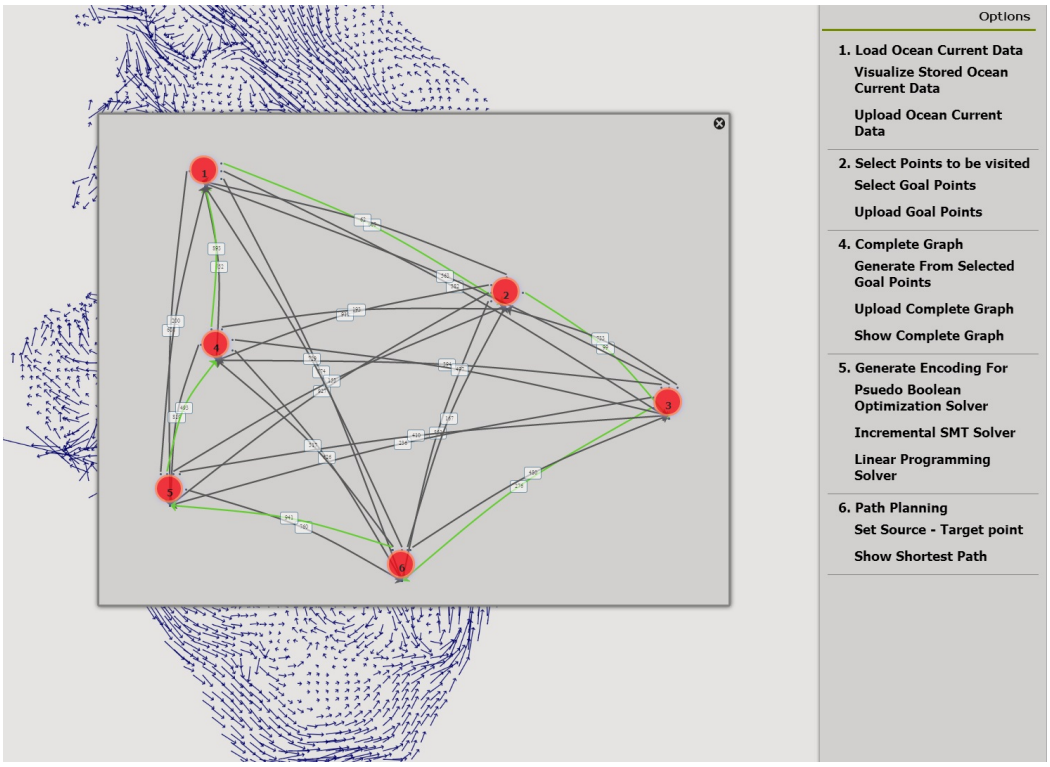

Figure 6.7: Ocean data visualization and goal points selection

is passed to the respective solver. Please see the previous chapters for the discussion of the encodings we implemented with their performance comparison.

When user selects the type of the solver, a back-end web server generates the corresponding encoding. At that time, the user can do any custom modification to the resulting file such as adding extra constraints. When a solver returns an optimal solution, the corresponding tour is extracted and visualized using the web interface (see Figure 6.7).

Chapter 7

Conclusion

We considered the mission planning problem for AUVs in the context of computing an optimal sequence of locations to be visited. As this is a variant of the NP-hard Travelling Salesman Problem, we used generic solvers such as Integer Llinear Programming, SAT/SMT and pseudo-Boolean optimization solvers to solve the core problem. The main practical reason for the choice of the solvers was our an emphasis on extendability by a wide variety of other constraints; from the theoretical point of view, we were interested on the behaviour of solvers on a set of instances coming from a well-defined real-life problem. In the course of this project, we built a software package for mission planning which we plan to make available to the community. A paper based on the practical part on our work has been accepted to the Journal of Ocean Technology; the work on theoretical side is still in progress.

Our performance comparison indicates that our instances were solvable by several solvers on up to 30 goal points, however only CPLEX solver could provide results within

an acceptable timeframe, solving instances on 50+ points. Thus, unless additional Boolean constraints dominate the size of the TSP instance, we suggest using MILP framework to encode the problem as well as the additional constraints (though we have not evaluated how much of the performance gain was due to the optimizations in CPLEX, as it is an industrial-strength solver). In particular, CPLEX solver outperformed pseudo-Boolean Optimization solvers, which in turn showed better results than running an SMT solver iteratively, for either Integer Linear Arithmetic or Undefined Function theory as an underlying theory.

Generally, performance of a solver can vary greatly with the type of an formulation of a problem; we have investigated whether there is a noticeable change in performance with different formulations of TSP and time dependent TSP. In our experiments, DL encoding gave the best performance and time dependent FGG encoding the worst, for the same number of goals.

From the solver performance analysis point of view, we were interested in the properties of instances that make them easier or harder for the solvers, as well as in determining whether our real-life data gives us instances resembling any of the easier-to-analyse synthetic data. We only did a few preliminary experiments, but so far our results seem to suggest that more symmetric data such as TSP instances coming from a Euclidean graph are harder for the solvers (CPLEX and PBO solvers) than the data with a lot of asymmetry, with random data being the easiest. Real-life data, surprisingly, also gave rise to easier instances, with performance of solvers being comparable between real-life and random data.

Bibliography

[AMP09] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania, Bounded model checking of software using smt solvers instead of sat solvers, International Journal on Software Tools for Technology Transfer **11** (2009), no. 1, 69–83.

[ARMS02] Fadi A Aloul, Arathi Ramani, Igor Markov, and Karem Sakallah, Pbs: a backtrack-search pseudo-boolean solver and optimizer, Proceedings of the 5th International Symposium on Theory and Applications of Satisfiability, 2002, pp. 346–353.

[Aro96] Sanjeev Arora, Polynomial time approximation schemes for euclidean tsp and other geometric problems, Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, IEEE, 1996, pp. 2–11.

[BB09] Robert Brummayer and Armin Biere, Boolector: An efficient smt solver for bit-vectors and arrays, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2009, pp. 174–177.

[BBR+06] Olivier Bailleux, Yacine Boufkhad, Olivier Roussel, et al., A translation of pseudo boolean constraints to sat., JSAT **2** (2006), no. 1-4, 191–200.

[Bek06] Tolga Bektas, The multiple traveling salesman problem: an overview of formulations and solution procedures, Omega **34** (2006), no. 3, 209–219.

[BH02] Endre Boros and Peter L Hammer, Pseudo-boolean optimization, Discrete applied mathematics **123** (2002), no. 1, 155–225.

[BHP09] Timo Berthold, Stefan Heinz, and Marc E Pfetsch, Solving pseudo-boolean problems with scip.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (eds.), Handbook of satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, February 2009.

[Bie08] Armin Biere, Picosat essentials., JSAT **4** (2008), no. 2-4, 75–97.

[BLM01] Per Bjesse, Tim Leonard, and Abdel Mokkedem, Finding bugs in an alpha microprocessor using satisfiability solvers, Computer Aided Verification, Springer, 2001, pp. 454–464.

[BNO+08] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio, The barcelogic smt solver, Computer Aided Verification, Springer, 2008, pp. 294–298.

[BTI11] Deepak Bhadauria, Onur Tekdas, and Volkan Isler, Robotic data mules for collecting data over sparse sensor fields, Journal of Field Robotics **28** (2011), no. 3, 388–404.

[Coo] William Cook, Concorde TSP solver, `http://www.tsp.gatech.edu/concorde/index.html`.

[Coo71] S.A. Cook, The complexity of theorem-proving procedures, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[DL91] Martin Desrochers and Gilbert Laporte, Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints, Oper. Res. Lett. **10** (1991), no. 1, 27–36.

[DLL62] Martin Davis, George Logemann, and Donald Loveland, A machine program for theorem-proving, Commun. ACM **5** (1962), no. 7, 394–397.

[DMB08] Leonardo De Moura and Nikolaj Bjørner, Z3: An efficient smt solver, Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.

[dMB09] Leonardo de Moura and Nikolaj Bjørner, Satisfiability modulo theories: An appetizer, Formal Methods: Foundations and Applications, Springer, 2009, pp. 23–36.

[DP60] Martin Davis and Hilary Putnam, A computing procedure for quantification theory, J. ACM **7** (1960), no. 3, 201–215.

[DPS] N Drucker, M Penn, and O Strichman, Cyclic routing of unmanned air vehicles, Tech. report.

[DST80] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan, Variations on the common subexpression problem, J. ACM **27** (1980), no. 4, 758–771.

[Eic13] Mike Eichhorn, Optimal routing strategies for autonomous underwater vehicles in time varying environment, Robotics and Autonomous Systems (2013).

[ES04] Niklas Eén and Niklas Sörensson, An extensible sat-solver, Theory and applications of satisfiability testing, Springer, 2004, pp. 502–518.

[FGG80] Kenneth R Fox, Bezalel Gavish, and Stephen C Graves, Technical notean n-constraint formulation of the (time-dependent) traveling salesman problem, Operations Research **28** (1980), no. 4, 1018–1021.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub, clasp: A conflict-driven answer set solver, Logic Programming and Nonmonotonic Reasoning, Springer, 2007, pp. 260–265.

[GP99] Luis Gouveia and Jose Manuel Pires, The asymmetric travelling salesman problem and a reformulation of the miller tucker zemlin constraints, European Journal of Operational Research **112** (1999), no. 1, 134 – 146.

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz, Boosting combinatorial search through randomization, Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence (Menlo Park, CA, USA), AAAI '98/IAAI '98, American Association for Artificial Intelligence, 1998, pp. 431–437.

[GSMT98] Carla P Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff, Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems, AIPS, 1998, pp. 208–213.

[GV95] Luis Gouveia and Stefan Voss, A classification of formulations for the (time-dependent) traveling salesman problem, European Journal of Operational Research **83** (1995), no. 1, 69 – 82.

[GVC95] Nabil Guerinik and Michel Van Caneghem, Solving crew scheduling problems by constraint programming, Principles and Practice of Constraint ProgrammingCP'95, Springer, 1995, pp. 481–498.

[Hel00] Keld Helsgaun, An effective implementation of the lin–kernighan traveling salesman heuristic, European Journal of Operational Research **126** (2000), no. 1, 106–130.

[HLW+08] Guoqi Han, Zhaoshi Lu, Zeliang Wang, James Helbig, Nancy Chen, and Brad De Young, Seasonal variability of the labrador current and shelf circulation off newfoundland, Journal of Geophysical Research: Oceans (1978–2012) **113** (2008), no. C10.

[HWB09] M. He, C. D. Williams, and R. Bachmayer, Simulations of an iterative mission planning procedure for flying gliders into strong ocean currents, 16th International Symposium on Unmanned Untethered Submersible Technology (UUST'09), 2009.

[IBM] CPLEX IBM, CPLEX Solver, `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[JDM11] Dejan Jovanović and Leonardo De Moura, Cutting to the chase solving linear integer arithmetic, Automated Deduction–CADE-23, Springer, 2011, pp. 338–353.

[Jua10] Juan Jose Miranda-Bront and Isabel Mendez-Diaz and Paula Zabala, An integer programming approach for the time-dependent TSP, Electronic Notes in Discrete Mathematics **36** (2010), no. 0, 351 – 358.

[JV83] Roy Jonker and Ton Volgenant, Transforming asymmetric into symmetric traveling salesman problems, Operations Research Letters **2** (1983), no. 4, 161–163.

[KS+92] Henry A Kautz, Bart Selman, et al., Planning as satisfiability, ECAI, vol. 92, 1992, pp. 359–363.

[LBP+10] Daniel Le Berre, Anne Parrain, et al., The sat4j library, release 2.2, system description, Journal on Satisfiability, Boolean Modeling and Computation **7** (2010), 59–64.

[LK73] Shen Lin and Brian W Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations research **21** (1973), no. 2, 498–516.

[Lue03] David G. Luenberger, Linear and nonlinear programming, Kluwer Academic Publ., Boston, Dordrecht, London, 2003.

[Mar09] Marques-Silva, Joao P. and Lynce, Ines and Malik, Sharad, Conflict-Driven Clause Learning SAT Solvers, ch. 4, pp. 131–153, vol. 185 of Biere et al. [BHvMW09], February 2009.

[Mit99] Joseph SB Mitchell, Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems, SIAM Journal on Computing **28** (1999), no. 4, 1298–1309.

[MMZ+01] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, Chaff: Engineering an efficient sat solver, Proceedings of the 38th annual Design Automation Conference, ACM, 2001, pp. 530–535.

[MSS96] J.P. Marques Silva and K.A. Sakallah, Grasp-a new search algorithm for satisfiability, Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on, 1996, pp. 220–227.

[MSS99] João P Marques-Silva and Karem A Sakallah, Grasp: A search algorithm for propositional satisfiability, Computers, IEEE Transactions on **48** (1999), no. 5, 506–521.

[MTZ60] C. E. Miller, A. W. Tucker, and R. A. Zemlin, Integer programming formulations of traveling salesman problems, Journal of the Association for Computing Machinery **7** (1960), 326–329.

[NO80] Greg Nelson and Derek C Oppen, Fast decision procedures based on congruence closure, Journal of the ACM (JACM) **27** (1980), no. 2, 356–364.

[NO07] Robert Nieuwenhuis and Albert Oliveras, Fast congruence closure and extensions, Information and Computation **205** (2007), no. 4, 557–580.

[OAL09] Temel Oncan, I. Kuban Altinel, and Gilbert Laporte, A comparative analysis of several asymmetric traveling salesman problem formulations, Computers and Operations Research **36** (2009), no. 3, 637 – 654.

[Opt93] CPLEX Optimization, Using the cplex callable library and cplex mixed integer library, CPLEX Optimization, Incline Village (1993).

[pbo] Pseudo boolean file format, `http://www.cril.univ-artois.fr/PB12/format.pdf`.

[Sab12] Sabharwal, Ashish and Samulowitz, Horst and Sellmann, Meinolf, Learning back-clauses in SAT, Theory and Applications of Satisfiability Testing–SAT 2012, Springer, 2012, pp. 498–499.

[SB09] Niklas Sörensson and Armin Biere, Minimizing learned clauses, Theory and Applications of Satisfiability Testing-SAT 2009, Springer, 2009, pp. 237–243.

[SBDL01] Aaron Stump, Clark W Barrett, David L Dill, and Jeremy Levitt, A decision procedure for an extensional theory of arrays, Logic in Computer Science, Symposium on, IEEE Computer Society, 2001, pp. 0029–0029.

[SD02] Hanif D Sherali and Patrick J Driscoll, On tightening the relaxations of miller-tucker-zemlin formulations for asymmetric traveling salesman problems, Operations Research **50** (2002), no. 4, 656–669.

[so04] SMT-LIB standards organization, The Satisfiability Modulo Theories Library, `http://smtlib.org/`, 2004, [Online SMT-LIB standards].

[Sur12] Pavel Surynek, A sat-based approach to cooperative path-finding using all-different constraints., SOCS, 2012.

[Tsa93] Edward Tsang, Foundations of constraint satisfaction, vol. 289, Academic press London, 1993.

[VKR+05] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter Corke, Data collection, storage, and retrieval with an underwater sensor network, Proceedings of the 3rd international conference on Embedded networked sensor systems, ACM, 2005, pp. 154–165.

[WPG05] I. Woodrow, A. Purry, C.and Mawby, and J. Goodwin, Autonomous auv mission planning and replanning - towards true autonomy, 14th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH, 2005.

[z3W] Z3 High-performance Solver, `http://z3.codeplex.com/`, [Z3 Web site].