Math 441 Discrete Optimization Learning Portfolio

Theo Diederichsen (86125424)

March 29, 2024

Contents

Proof of Convex Set 2
Review of Math 340 Term Project 4
Interview with RBC Global Asset Management Quantitative Group 6
Using Google OR-Tools to Solve Optimization Problems 7
TSP: Comparing MTZ & DFJ 12
Literature Review of Compressive Sensing in Medical Imaging 15

List of Figures

1 Convex Set 2

List of Tables

Proof of Convex Set

Theorem 10.1 completed from the textbook "*Linear Programming Foundations and Extensions*", Fifth Edition, by Robert J. Vanderbei [1].

A set C is convex if and only if it contains all convex combinations of points in C [1].

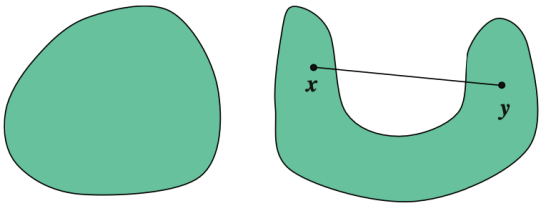


Figure 1: Convex Set

Proof. Proof by induction.

Let C be a convex set. By definition, C contains all convex combinations of pairs of points in C. Then $\forall z_i \in C,\ i \in \mathbb{N}$, the connection of the z_i's are all in C.

Base case step[2]: Fix z_1, z_2, and z_3 in C and consider

$$z = t_1 z_1 + t_2 z_2 + t_3 z_3$$

where $t_j \geqslant 0$ for each j and $\sum_{j=1}^{3} t_j = 1$. If any of the t_j's vanish, then z is really just a convex combination of two points and so belongs to C. Hence, suppose that each of the t_j's is strictly positive. Rewrite z as follows:

$$\begin{aligned} z &= (1 - t_3)\left(\frac{t_1}{1 - t_3} z_1 + \frac{t_2}{1 - t_3} z_2\right) + t_3 z_3 \\ &= (1 - t_3)\left(\frac{t_1}{t_1 + t_2} z_1 + \frac{t_2}{t_1 + t_2} z_2\right) + t_3 z_3 \end{aligned}$$

Since C contains all convex combinations of pairs of points, it follows that

$$\frac{t_1}{t_1 + t_2} z_1 + \frac{t_2}{t_1 + t_2} z_2 \in C.$$

Now, since z is a convex combination of the two points $\frac{t_1}{t_1+t_2} z_1 + \frac{t_2}{t_1+t_2} z_2$ and z_3, both of which belong to C, it follows that z is in C.

[1] Reading through the Vanderbei textbook, and many mathematical textbooks for that matter, I have found many proofs 'left for the reader to solve'. I though I would take the opportunity to finish a proof of a convex set which was left incomplete (Theorem 10.1). It is particularly neat having a clear mathematical definition of something that is usually very obvious geometrically.

[2] Copied directly from the Vanderbei textbook.

Induction step:

Assume that all connections of $z_n \in C$, then all the connections of the $z_{n+1} \in C$.

Let $z^+ = t_1 z_1 + t_2 z_2 + \cdots + t_{n+1} z_{n+1}$ where $t_i \geqslant 0$, and $\sum_{i=1}^{n+1} t_i = 1$. If any of the t_i's vanish, then z is really just a convex combination of n points and thus belongs to C.

Hence, suppose that $\forall i, t_i > 0$. Then we get:

$$z^+ = (1 - t_{n+1}) \left(\frac{t_1}{1 - t_{n+1}} z_1 + \frac{t_2}{1 - t_{n+1}} z_2 \cdots \frac{t_n}{1 - t_{n+1}} z_n \right) + t_{n+1} z_{n+1}$$

$$= (1 - t_{n+1}) \left(\frac{t_1}{t_1 + t_2 + \cdots + t_n} z_1 + \frac{t_2}{t_1 + t_2 + \cdots + t_n} z_2 \cdots + \frac{t_{n-1}}{t_1 + t_2 + \cdots + t_n} z_n \right) + t_{n+1} z_{n+1}$$

Since Since C contains all convex combinations of n points, it follows that:

$$\frac{t_1}{t_1 + t_2 + \cdots + t_n} z_1 + \frac{t_2}{t_1 + t_2 + \cdots + t_n} z_2 \cdots + \frac{t_{n-1}}{t_1 + t_2 + \cdots + t_n} z_n \in C$$

Now, since z^+ is a convex combination of the n points $\frac{t_1}{t_1+t_2+\cdots+t_n} z_1 + \frac{t_2}{t_1+t_2+\cdots+t_n} z_2 \cdots + \frac{t_{n-1}}{t_1+t_2+\cdots+t_n} z_n$ and z_{n+1} both of which belong to C, it follows that $z^+ \in C$.

□

Review of Math 340 Term Project

The research paper *"Optimizing Financial Resource Allocation to Maximize the University of British Columbia's Student Satisfaction Index"* [2], seeks the possibilities of improving the well-being of UBC students within specified financial constraints. It uses linear programming to identify an optimal solution to the problem. The paper outlines the data collection process and provides insight into its implementation[3]. This critique will specifically center on evaluating the application of linear programming, while avoiding a broad assessment of the paper's limitations or overall structure.

While the paper's overall approach is logical, the lack of data used in constructing the A-matrix significantly compromises the reliability of the outcomes. The objective is to optimize a "Student Satisfaction Index," with coefficients representing the proportion of each category contributing to student satisfaction, while the constraint is the total tuition collected by UBC. Consequently, the A-matrix signifies the cost of a unit increase in satisfaction for a specified category. The paper relies on only three years of survey results to derive these values, resulting in an very large variance in the A-matrix values due to only two observed year-over-year changes. Moreover, two categories had to be excluded due to a negative correlation between funding and student satisfaction, further diminishing the credibility of the findings.

The analysis in the paper lacks clarity regarding its perspective. It should distinctly establish whether it approaches the problem from the standpoint of the students or the university. While maximizing a student satisfaction index might suggest prioritizing various services within a tuition constraint from the student's viewpoint, it could also be interpreted as the university seeking the most favorable review without increasing expenditure on services. Once a stance is chosen, it is important to look into the alternative perspective to ensure a comprehensive understanding of both parties involved. Therefore, a thorough analysis of the dual perspective will contribute to a more insightful analysis of the resource allocation optimization.

The index definition implies that all decision variables have a lower bound of zero and an upper bound of a hundred. The paper's strategy involved applying the standard form of linear programming and utilizing the Google Linear Optimization Solver (GLOP) from OR-tools to obtain an optimal solution. A more efficient approach would have entailed employing the general form of linear programming. Through this, the modified simplex method could have been applied, removing the necessity to reframe the problem in standard form [3]. This would not only enhance efficiency but also contribute to a clearer process.

The general linear programming takes the form of:

Maximize or Minimize:

$$c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to:

$$\begin{aligned} b_{11} \leqslant a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leqslant b_{12} \\ b_{21} \leqslant a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leqslant b_{22} \\ \vdots \\ b_{m1} \leqslant a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leqslant b_{m2} \\ d_1 \leqslant x_1, x_2, \ldots, x_n \leqslant d_2 \end{aligned}$$

Arguably, the most significant assumption made was that the problem follows a linear form. While

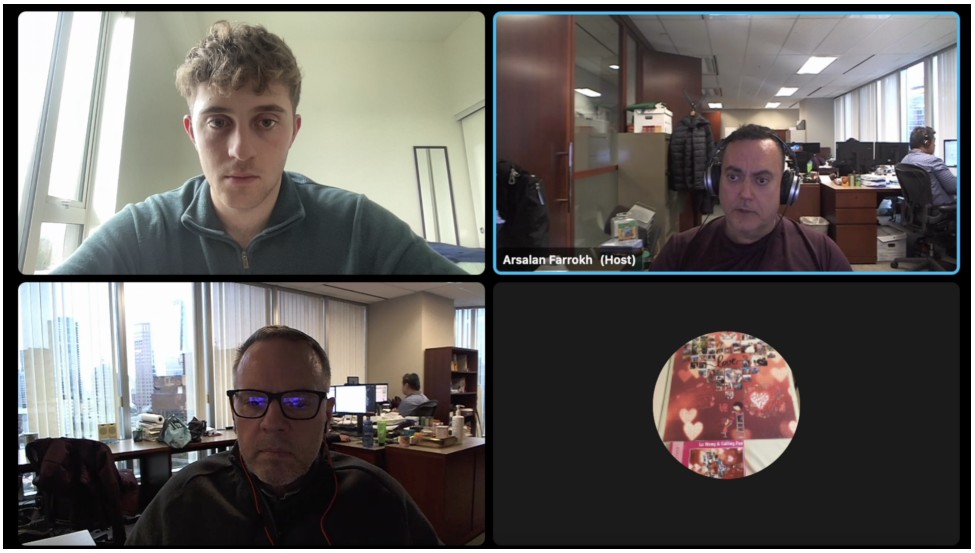
[3] As this was an optional project, this only scratched the surface of what could be a very in-depth and involved analysis. We were aware of this going into the project, however the lack of data available to us, and perhaps lack of incentive resulted in an analysis with many holes in it.

this may hold true for minor shifts in funding, substantial resource redistribution would likely fit to a quadratic model. It becomes apparent that the removal of a particular service, even if impacting only a small proportion of students directly, can have an indirect and extensive impact on a larger student population, resulting in diminished satisfaction ratings for the university. While the paper attempts to address this by imposing a minimum satisfaction requirement for all categories, the proposed magnitude of change still surpasses the constraints outlined.

In summary, the paper could benefit from fine-tuning to produce a more meaningful analysis. By addressing the limitations related to data adequacy and linear programming application, the analysis shows opportunities for improvement in the methodology. This shows the importance of clearly defining perspectives and suggests options for redefining the optimization process to improve the robustness and validity of the paper's findings.

Interview with RBC Global Asset Management Quantitative Group

After finishing an internship at RBC Global Asset Management (GAM), I had the opportunity to explore diverse applications of mathematical models. Interested by the use of linear programming (LP), I decided to interview the quantitative investment department [4]. During this process, I connected with Rob Menning, CFA, Vice President of the Quantitative Research Group, Arsalan Farrokh, Ph.D., CFA, FRM, a Quantitative Investment Analyst, and Lu Wang, CFA, FRM, PRM, a Quantitative Analyst Developer. They provided insights and answered some of my questions about the application of LP.



Using Google OR-Tools to Solve Optimization Problems

For best functionality, use the .ipynb file.

OR-Tools is a collection of tools developed by Google for optimization problems, including linear programming, mixed-integer programming, and constraint programming [5].

The first step is to install the OR-Tools package. It will produce a list of all items installed.

```
! pip install ortools
```

Let us also import numpy, it will be useful.

```
import numpy as np
```

Importing Modules

There are various different modules for optimization problems in OR-Tools. Each module is designed to address specific types of optimization problems and provides algorithms and tools tailored to those problems. Some of the most commonly used ones are:

1. **ortools.constraint_solver**: This module provides functionalities for solving constraint satisfaction problems, which involve finding solutions that satisfy a set of constraints.

2. **ortools.graph**: This module offers algorithms and data structures for solving graph-related problems, such as finding shortest paths, minimum spanning trees, and maximum flow problems.

3. **ortools.linear_solver**: This module provides functionality for solving linear programming problems.

The full list can be found in the OR-Tools Python Reference[the OR-Tools Python Reference].

Let us work with **pywraplp** component from the **ortools.linear_solver** module. The pywraplp component provides classes and methods for constructing linear programming models, defining variables and constraints, setting objective functions, and solving the resulting optimization problems.

```
from ortools.linear_solver import pywraplp
```

We can check what has been imported by calling **pywraplp.__dir__()**.

Note that the most useful classes here are **"Objective", "Constraint", "Variable"**, and **"Solver"**

Creating a 'GLOP' solver

GLOP (Google Linear Optimization Package) is Google's linear programming solver. It uses the simplex algorithm to solve linear programming problems. There are other solver libraries that may be used for different

1. **CBC** : The Coin-OR branch and cut (CBC) solver is an open-source linear programming solver that uses a combination of branch-and-bound and cutting plane methods.

2. **CLP** : The Coin-OR linear programming (CLP) solver is linear programming solver from the Coin-OR project. It also uses the simplex algorithm.

3. **SCIP** : SCIP (Solving Constraint Integer Programs) is a mixed-integer programming solver. While primarily designed for mixed-integer programming, it can also handle linear programming problems.

4. **Gurobi** : Gurobi is a commercial optimization solver known for its performance on linear programming and mixed-integer programming problems. It may be integrated with OR-Tools if you have access to a Gurobi license.

5. **CPLEX** : IBM's CPLEX Optimizer is another commercial optimization solver widely used for linear programming and mixed-integer programming. Similar to Gurobi, it may be integrated with OR-Tools if you have access to a CPLEX license.

We will be using this to solve all of our optimization problems.

```
solver = pywraplp.Solver.CreateSolver("GLOP")
```

We can also check what is in the **GLOP solver**, by calling **solver.__dir__()**.

We note the most useful methods here to be "**Objective", "NumVar", "Constraint", "NumConstraints","Add","Minimize","Maximize", "LookupVariable"**, and **"LookupConstraint"** .

Example 1

Now we have all the tools necessary to begin working with an example. Consider the linear programming problem given as an example by google:

$$\max 3x + 4y$$
$$x + 2y \leqslant 14$$
$$3x - y \geqslant 0$$
$$x - y \leqslant 2$$

Creating Variables

Let us create the variables

```
x = solver.NumVar(-solver.infinity(), solver.infinity(), "x")
y = solver.NumVar(-solver.infinity(), solver.infinity(), "y")
```

We can check variable bounds

```
print(f"lower and upper bounds of x: {x.lb()}, {x.ub()}")
print(f"lower and upper bounds of y: {y.lb()}, {y.ub()}")
```

Let us also check what we functions we can call on a variable, by calling **x.__dir__()**. We note useful variable methods **"lb","ub","SetBounds"** , and **"solution_value"** .

Constraints

We can now add constraints. We use variable objects as if they are numbers.

```
solver.Add(x + 2 * y <= 14.0)

solver.Add(3 * x - y >= 0.0)

solver.Add(x - y <= 2.0)
```

Objective Function

Finally, we define an objective function.

```
objective = solver.Maximize(3 * x + 4 * y)
```

Starting the Solver

After constructing a linear programming model by defining variables, constraints, and the objective function, we are ready to use 'solver.Solve()'. 'solver.Solve()' is the method used to solve the optimization problem. It applies the simplex method, as we are using a **GLOP** solver in this case.

It returns a status indicating whether a solution was found and, if so, the quality of that solution (e.g., optimal, feasible, infeasible, unbounded).

```
status = solver.Solve()
```

Checking the Solution

We can check if our solution is optimal by using **pywraplp.Solver.OPTIMAL** against **status**.

To check the Optimal value, we use **solver.Objective().Value():.0f**. To adjust the decimal precision, change **.0f** part to as many decimal places as you wish.

```
if status == pywraplp.Solver.OPTIMAL:
    print(f"Optimal solution is: x={x.solution_value():.0f}, y={y.solution_value():.0f}")
    print(f"Optimal value is: {solver.Objective().Value():.0f}")
```

Using Matrix Notation

Let us see a simple example using matrix notation:

$$\max x_1 + 2x_2$$

$$2x_1 + 3x_2 \leqslant 6$$
$$-4x_1 + x_2 \leqslant 10$$

We follow a very similar procedure, writing it all in one function

```
from ortools.linear_solver import pywraplp

# Define the function
def Solver2():
    data_obj = [1,2]

    data_A = [
        [2,3],
        [-4,1]
    ]

    data_b = [6,10]

    #Create a GLOP solver
    solver = pywraplp.Solver.CreateSolver('GLOP')
    if not solver:
        return

    # define our variables
    xs = ['x1','x2']
    vars = [solver.NumVar(0.0, solver.infinity(), x) for x in xs]

    # constraints, one per inequality (equality)
    constraints = []
    for i, b in enumerate(data_b):
        constraints.append(solver.Constraint(-solver.infinity(),b))
        for j, d in enumerate(data_A[i]):
            constraints[i].SetCoefficient(vars[j], d)

    # objective: maximization - so SetMaximization
    objective = solver.Objective()
    for j, d in enumerate(data_obj):
        objective.SetCoefficient(vars[j], d)
    objective.SetMaximization()

    status = solver.Solve()

    # Check that the problem has an optimal solution
    if status != solver.OPTIMAL:
        print('The problem does not have an optimal solution!')
        if status == solver.FEASIBLE:
            print('A potentially suboptimal solution was found.')
        else:
```

```
            print('The solver could not solve the problem.')
            exit(1)

    # Display solution
    solution = [0] * len(data_obj)
    for i, var in enumerate(vars):
        print(f'{xs[i]} : {var.solution_value():.1f}')
    print(f'\nOptimal value: {objective.Value():.1f}')

Solver2()
```

TSP: Comparing MTZ & DFJ

The Traveling Salesperson Problem (TSP) is a classic routing problem where we aim to find the shortest closed path connecting N locations. Consider the complete graph $G = (V, E)$, with nodes enumerated as $i = 0, \ldots, N-1$.

The TSP can be formulated as an integer programming problem [6]. Several formulations exist, with two notable ones being the Miller–Tucker–Zemlin (MTZ) formulation and the Dantzig–Fulkerson–Johnson (DFJ) formulation. While the DFJ formulation is stronger, the MTZ formulation remains useful in certain scenarios [7].

Let x_{ij} be 1 if the path includes the edge from node i to j, and 0 otherwise. The distance (or cost) to travel from node i to j is denoted as c_{ij}. The objective function is the total cost of the path:

$$\min \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j \neq i}}^{n} c_{ij} x_{ij}$$

Without further constraints, the variables x_{ij} would effectively range over all subsets of the set of edges, which is far from the sets of edges in a tour. This allows for a trivial minimum where all $x_{ij} = 0$. Hence, both formulations include constraints ensuring that at each vertex, there is exactly one incoming edge and one outgoing edge. These constraints can be expressed as $2n$ linear equations:

One edge from each node in the path:

$$\sum_{j, j \neq i} x_{ij} = 1, \text{ for each } i$$

One edge to each node in the path:

$$\sum_{i, i \neq j} x_{ij} = 1, \text{ for each } j$$

This approach still permits solutions that violate the global requirement of having only one tour that visits all vertices. The selected edges could potentially form multiple tours, each visiting only a subset of the vertices. The MTZ and DFJ formulations are different in how they implement this final requirement as linear constraints.

Miller–Tucker–Zemlin approach

In addition to the x_{ij} variables mentioned earlier, there's another set of variables we denote u_i for each city $i = 1, \ldots, n$. These u_i variables keep track of the order in which cities are visited, starting from city 1. If $u_i < u_j$, it means city i is visited before city j. To find suitable values for the u_i variables for a given tour, we can set u_i to the number of edges traveled along that tour when going from city 1 to city i.

In linear programming, it's better to use non-strict inequalities rather than strict ones. So, we want to make sure that if there's an edge from city i to city j ($x_{ij} = 1$), then u_j should be at least $u_i + 1$. Simply requiring $u_j \geqslant u_i + x_{ij}$ wouldn't work because it would also force $u_j \geqslant u_i$ when $x_{ij} = 0$. Instead, the MTZ approach uses $n(n-1)$ linear constraints:

$$u_i - u_j + 1 \leqslant (n-1)(1 - x_{ij})$$

for all distinct cities $i, j \in \{2, \ldots, n\}$. The constant term $(n-1)$ provides enough flexibility so that when $x_{ij} = 0$, there's no requirement for u_j in relation to u_i, as the inequality will always be at most $(n-1)$.

The u_i variables ensure that a single tour visits all cities by increasing by 1 for each step along the tour, with a decrease only allowed where the tour passes through city 1. If a tour doesn't pass through city 1, it would violate this constraint, so the only way to satisfy it is for the tour passing through city 1 to also pass through all other cities.

So we get the additional constraints:

$$u_i - u_j + 1 \leqslant (n-1)(1 - x_{ij}), \text{ for each } 2 \leqslant i \neq j \leqslant n$$
$$2 \leqslant u_i \leqslant n, \text{ for each } 2 \leqslant i \leqslant n$$

Dantzig–Fulkerson–Johnson approach

The final constraint in the DFJ formulation is known as a subtour elimination" constraint. Its purpose is to prevent any subset of cities from forming its own smaller tour within the solution. This ensures that the solution returned represents a single tour that visits all cities, rather than a combination of smaller tours. However, implementing these constraints directly could result in an enormous number of constraints, which is not practical. Therefore, in practice, this issue is addressed using a technique called "branch and cut." Branch and cut is a method that combines elements of both branch and bound and cutting plane methods to efficiently solve optimization problems like the TSP. It helps to iteratively refine the solution space by adding constraints cutting planes to eliminate unwanted regions while exploring different branches of the solution space. This approach helps to efficiently find an optimal or near-optimal solution to the TSP.

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leqslant |Q| - 1, \text{ for all } Q \subsetneq \{1, \ldots, n\}, |Q| \geqslant 2$$

Research studies have shown that the DFJ formulation tends to offer improved performance in terms of solution time compared to the MTZ formulation. For example, one study by Applegate, Bixby, Chvátal, and Cook found that the DFJ formulation outperformed the MTZ formulation on a set of benchmark instances from the TSPLIB library [8]. They reported that the DFJ formulation solved instances faster on average, with some instances showing significant speed improvements, even as much as %50 faster.

Additionally, other research papers and computational experiments have provided similar findings, showing that the DFJ formulation can lead to faster solution times compared to the MTZ formulation for a wide range of TSP instances.

Overall, solving the TSP, the DFJ approach is often seen as better than the MTZ approach for a few key reasons. Firstly, DFJ is stricter in its approach, meaning it sets stronger rules to ensure the solution is more accurate. This can help algorithms find better answers more quickly. Additionally, DFJ is simpler to understand and use compared to MTZ, especially for complex TSPs. This simplicity makes it easier to work with and implement. Also, DFJ tends to work faster than MTZ, particularly for larger or more challenging TSP instances. One significant advantage of DFJ is that it avoids the need for additional steps to prevent the solution from breaking into smaller tours. This not only saves time but also strengthens the

solution overall. While both methods have their strengths, DFJ is often preferred for its stricter, simpler, and faster approach to solving the TSP. However, the choice between the two methods may depend on the specific problem being tackled and the computational resources available.

Literature Review of Compressive Sensing in Medical Imaging

The article *Compressive Sensing in Medical Imaging* [9] explores how compressive sensing (CS) can improve medical imaging by generating high-quality images with less data. It discusses the importance of CS, examines its use in x-ray CT and MRI, and looks at its impact on imaging technology. Medical imaging costs drive interest in CS, supported by the controllable radiation source and indirect nature of raw data in imaging. CS reduces data needs in MRI, benefiting dynamic and quantitative imaging by shortening scan times. Similarly in CT, CS can reduce doses, address motion artifacts, and create new scan designs, though adapting it to already used technologies may not be as effective, emphasizing practical evaluations over expecting drastic changes.

MRI uses the magnetic resonance effect to image hydrogen nuclei, enabling various types of imaging. It encodes spatial information through magnetic field gradients, often mimicking a fourier basis function, before emitting location-encoded radiofrequency radiation for detection. CT imaging uses x-rays and advanced hardware to produce high-resolution images, with applications from cardiac imaging to tomosynthesis[4]. Research in both MRI and CT focuses on image reconstruction, with iterative methods gaining attention in CT for exploiting image gradients, while hardware advancements aim to improve image quality and reduce radiation exposure.

CS has potential in reconstructing images in CT scans. It works by taking advantage of the fact that many parts of the body in CT scans are similar and only change abruptly at certain points. CS uses mathematical optimization models, like total variation, to help make images clearer. It's like putting together a puzzle with missing pieces, but you can still tell what the picture is supposed to be. CS also tries to make the process faster and use less data, which can be tricky because CT machines sometimes give inconsistent results. Scientists are working to make CS work better for CT scans by finding optimizing algorithms to make it more reliable and easier to use for physicians.

Before discussing potential solutions, let's revisit the fundamental imaging model for X-ray CT and MRI:

$$\mathbf{g} = \mathbf{A}\mathbf{h}$$

where A stands for either X, when modeling CT, or F when modeling MRI. The issue of data inconsistency means that only under the conditions of ideal noiseless data used for obtaining a compressive sensing guarantee is it possible to solve. The proposed optimization strategy for a compressive sensing guarantee was to minimize the total variation (TV) of the image while aligning the image estimate with available projection data:

$$\mathbf{h}^* = \text{argmin}_{\mathbf{h}} \text{TV}(\mathbf{h}) \quad \text{subject to} \quad \mathbf{X}\mathbf{h} = \mathbf{g}.$$

For real data or realistic simulations the data constraint needs to be relaxed. One such relaxation is:

$$\mathbf{h}^* = \text{argmin}_{\mathbf{h}} \text{TV}(\mathbf{h}) \quad \text{subject to} \quad D(\mathbf{X}\mathbf{h}, \mathbf{g}) \leq \epsilon$$

[4]Tomosynthesis, also known as 3D mammography, is an advanced imaging technique used primarily for breast imaging. It works by taking multiple X-ray images of the breast from different angles and then reconstructing them into a three-dimensional image. This approach allows radiologists to examine the breast tissue layer by layer, rather than as a single flat image, which can improve the detection of abnormalities, particularly in dense breast tissue. It can improve the detection of breast cancer by providing clearer images and reducing the likelihood of false positives or false negatives. [10]

where ϵ is a parameter controlling the allowable level of data discrepancy. In the CS for CT literature, the most common form of optimization for exploiting gradient magnitude image[5] sparsity is as unconstrained minimization:

$$\mathbf{h}^* = \text{argmin}_{\mathbf{h}} \{D(\mathbf{g}, \mathbf{X}\mathbf{h}) + \gamma \text{TV}(\mathbf{h})\}$$

which is essentially the same optimization arrived in considering edge-preserving regularization. The picture for edge-preserving regularization is that there is a trade-off between data fidelity and image regularity. Increasing γ leads to greater regularity at the cost of losing data fidelity.

These equations can be reformulated as linear programming problems and addressed by solving an underdetermined system of equations[6]. Algorithms like compressive sensing capitalize on image sparsity, allowing meaningful images to be reconstructed from fewer measurements than usual. They achieve this by iteratively minimizing an objective function, balancing fidelity to available data with desired image properties. Through this iterative adjustment based on measurements and prior knowledge, these algorithms produce high-quality images while solving the underdetermined system, promising faster scans, decreased radiation exposure, and potentially reduced costs in medical imaging.

CS techniques show promise for X-ray CT image reconstruction, leveraging sparsity and optimization models. Challenges include data inconsistencies, necessitating relaxation of constraints. Conditioning of the X matrix impacts image sensitivity and solver convergence rates. Ongoing efforts aim to optimize algorithms and develop practical methods tailored to CT imaging intricacies.

[5] A gradient magnitude image represents the intensity of changes in pixel values across an image, providing information about the rate of change of intensity. It is derived from the gradient of an image, which captures the direction and magnitude of intensity changes. The gradient magnitude image highlights edges and boundaries in an image, making it useful in various image processing tasks like edge detection and feature extraction. [11]

[6] An undetermined system of equations is a set of equations with more variables than equations, resulting in multiple possible solutions. It's like having too few constraints to uniquely determine the values of all variables, leaving some freedom in choosing solutions. This situation often arises in problems where there are fewer restrictions or conditions than unknowns, leading to infinite or multiple valid solutions.[12]

References

[1] ROBERT J. VANDERBEI "*Linear Programming Foundations and Extensions*", Fifth Edition.

[2] THEO DIEDERICHSEN, ROBIN MATHESON *Optimizing Financial Resource Allocation to Maximize the University of British Columbia's Student Satisfaction Index.*

[3] YUANLONG RUAN, 2023 *Lecture Notes*.

[4] ARSALAN FARROKH, ROB MENNING, LU WANG

[5] Google OR-Tools Website

[6] C.H. PAPADIMITRIOU; K. STEIGLITZ "*Combinatorial optimization: algorithms and complexity*", Mineola, NY: Dover.

[7] MARK VELEDNITSKY "*Short combinatorial proof that the DFJ polytope is contained in the MTZ polytope for the Asymmetric Traveling Salesman Problem*".

[8] DAVID APPLEGATE, ROBERT BIXBY, VAŠEK CHVÁTAL, WILLIAM COOK "*The Traveling Salesman Problem: A Computational Study*".

[9] CHRISTIAN GRAFF, EMIL SIDKY "*Compressive Sensing in Medical Imaging*".

[10] JULIAN KLEINKNECHT, ANCA CIUREA, CRISTIANA CIORTEA "*Pros and cons for breast cancer screening with tomosynthesis – a review of the literature*".

[11] ADRIAN ROSEBROCK mage Gradients with OpenCV

[12] DIRK COLBRY Underdetermined Systems

Pros and cons for breast cancer screening with tomosynthesis – a review of the literature