NORTHWESTERN UNIVERSITY

# Making Difference with Optimization and Big Data: Topics in Power Grid Visualization, Airline Fleet Assignment and Sports Play Retrieval

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Mingyang Di

EVANSTON, ILLINOIS

March 2017

# ABSTRACT

Making Difference with Optimization and Big Data: Topics in Power Grid Visualization, Airline Fleet Assignment and Sports Play Retrieval

Mingyang Di

As the title suggests, this dissertation is composed of three major topics. The first two are optimization problems focusing on developing effective solution methodologies, while for the last topic we present a large-scale information retrieval system in the domain of sports. With the algorithms and frameworks developed in this dissertation, we achieve a significant advancement in solving large-scale optimization problems with big data techniques, and in multi-agent sports play retrieval via distributed computing and learning to rank.

The operations in electric power control centers play a crucial role in ensuring the integrity of the nation's electric grid and present formidable challenges to human operators. One of the primary challenges of information display for a power transmission control room application is the clutter from displaying too much information in too small of a display space. Thus, the task to optimize the layout of visual elements consisting of substations and transmission lines on the display interface is of vital importance. To this end, algorithms using several optimization techniques including Lagrangian relaxation and progressive hedging (PH) are proposed in the

second chapter to make the interface less cluttered subject to human perceptual and cognitive capabilities. We conduct extensive computational studies to evaluate and compare the developed algorithms, and report our findings based on a real-world power grid in the U.S.

We then move on to the airline fleet assignment problem. Since in recent years airlines around the world continue to face increasing capital and operational costs, they have been forced to cut costs and uphold revenue by utilizing their equipment capacity more efficiently to accommodate passenger demand. This is generally known as the *fleet assignment problem*, which deals with assigning aircraft of different capacities to the scheduled flight legs based on availabilities, costs, potential revenue and itinerary-based passenger demand. In order to address the high level of uncertainty in the market demand when the fleeting decisions are made and to capture the network effects (i.e., spill and recapture) for a more accurate estimate of passenger flow, we present a two-stage stochastic model which incorporates an attractiveness-based spill and recapture framework in the third chapter. This model considers spill and recapture based on passenger utility from itineraries. Solution approaches based on a distributed framework, namely MapReduce, are developed to reduce the computational time, and numerical results are reported using real data from a medium-size airline to evaluate and compare the proposed procedures.

Finally, in the last chapter, we showcase a streaming-based retrieval system that can quickly find the most relevant basketball plays given an input query. The idea was inspired by the recent explosive growth of sports tracking data and the more and more important role sports analytics is playing in professional leagues. To search through a large number of games at an interactive speed, our system is built upon a distributed framework so that each query-result pair is evaluated in parallel. We also propose a pairwise learning to rank approach to improve

search ranking based on users' clickthrough behavior. The similarity metric in training the rank function is based on automatically learnt features from a convolutional neural network (CNN)-based autoencoder. In the end, we validate the effectiveness of our learning to rank approach by demonstrating rank quality in a user study.

This work was supported by the New York State Energy Research and Development, Sabre Airline Solutions and STATS LLC.

## Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Diego Klabjan for his guidance, support and patience throughout my Ph.D. study at Northwestern. He is truly a master in advising students, conducting research, and teaching, and more than anything, he is a role model of being so humble even after having achieved so much. It is indeed my privilege to work with him. For everything you've done for me, Dr. Klabjan, I thank you. Also, I would like to thank all the committee members, Dr. Karen Smilowitz and Dr. Pablo Luis Durango-Cohen, for their help and valuable advice for my dissertation, coursework and career pursuit.

Many thanks to my friends in Evanston for making my life here so enjoyable. I thank Dr. Kuo-Ling Huang, Liu Liu, Long Sha, Mark Harmon and Dr. Patrick Lucey for their technical and emotional support - whenever I need it. And many, many others in the department of Industrial Engineering and Management Sciences, Jo Ann Yablonka, Agnes Kaminski, Victoria Richmond, Johnathan Gaetz, ... Thank you all for the kindness to me.

Finally, and most importantly, I would like to thank my wife Qingyu. Her support, encouragement, and unwavering love were undeniably the bedrock upon which the past five years of my life have been built. Her tolerance of my occasional reckless moods is a testament in itself of her unyielding devotion and love. Without her, I should not be the person I am now. Also, I thank my parents for their faith in me and accepting my absence from home all these years.

# Table of Contents

CHAPTER 1

# Introduction

This dissertation is concerned with three topics in the areas of information display, airline scheduling and management, and sports analytics. In the first two, we develop optimization algorithms and heuristics to solve large-scale mixed-integer programs (MIP) and stochastic mixed-integer programs (SMIP), along with implementation aspects and applications of big data techniques in solving optimization problems. The last chapter presents a large-scale information retrieval system in the domain of sports. We achieve real time high quality retrieval via a distributed in-memory architecture and a *key-value* database model tailored for our retrieval task. Pairwise learning to rank and deep learning methods are also leveraged to improve search ranking based on users' clickthrough feedback.

As the reliability of the electric power grid is largely in the hands of human operators who manage it at control centers, an effective way to prevent possible disruptions to the supply of electric power is to design a display interface to allow human operators to process vast amount of information rapidly and reliably. One of the primary challenges associated with this task is the clutter from displaying too much information in too small of a display space. And it is further aggravated by the future need to broaden the views to also encompass neighboring control areas and display predicted values. Therefore, in this dissertation, we propose a multi-objective mixed-integer model to rearrange the layout of visual elements consisting of substations and transmission lines so that the display is less cluttered subject to human perceptual and cognitive capabilities. The entire concept relies on the notion of an embedded orthogonal grid in which

the edges consist of only vertical and horizontal lines. The intersections of edges are referred to as nodes, and they correspond to possible locations of substations.

Since the proposed model cannot be directly solved by any commercial solver, we develop two network partitioning-based algorithms - a Lagrangian relaxation algorithm and a progressive hedging (PH) algorithm - to decompose the global network into smaller regions and then iteratively solve the subproblem in each region by readjusting penalties for corresponding solutions. As to be elaborated, although the two algorithms are based on the same (decomposed) network, they rely on completely different assumptions to handle the problem at the boundaries. In the Lagrangian relaxation-based approach, we assume that boundary edges belong to neither the region they emanate from nor the region they inject into, and relax all the constraints pertaining to these edges. In the PH-based approach however, we assume the contrary - that is, boundary edges belong to both of these regions - and obtain two different flow values for each boundary edge. By updating the penalty terms in every iteration, the PH-based algorithm gradually equalizes the two values. We also develop a sequential routing heuristic to route transmission lines one by one based on a revised shortest path algorithm, and conduct extensive computational experiments to evaluate and compare the three proposed solution methodologies. Our findings show that the two network partitioning-based iterative algorithms deliver similar high quality solutions within 5 to 10 hours, while the sequential routing approach gives a preliminary result of much lower quality in a much shorter time. The comparison is indeed a trade-off between computational resources and the quality of the solution.

We then move on to the airline fleet assignment problem, which refers to the assignment of aircraft types, each having a different capacity, to the scheduled flights based on availabilities, costs, potential revenue and itinerary-based passenger demand. Due to the large number of

scheduled flights and the dependency of other airline operations (e.g., crew scheduling), making fleet assignment decision is never an easy task. In this dissertation, we address the major drawbacks of the previous fleet assignment models (FAM) by presenting a two-stage stochastic model that considers potential demand scenarios and attractiveness-based spill and recapture effects [71]. The first stage is a basic fleet assignment model which decides the assignment of fleet types to flight legs, while the second stage finds the optimal passenger flow on available itineraries for each scenario based on the assigned capacities and the demand estimated for each itinerary.

We develop three MapReduce-based solution approaches to solve scenario-dependent subproblems in parallel to reduce the computational time. MapReduce is a distributed framework that processes parallelizable applications involving big data; however in this dissertation, we leverage it to solve a large-scale stochastic program. The first two algorithms are based on Benders decomposition, in which all the second-stage subproblems are solved in parallel and the associated Benders cuts are generated separately. The third approach employs progressive hedging and works similarly by solving each scenario-dependent deterministic case simultaneously. Unlike traditional PH, we solve each scenario via a Benders decomposition algorithm and add the Benders cuts generated in previous iterations as extra constraints to the next iteration. We evaluate and compare these algorithms using data from a medium-size airline, and our computational results establish feasibility in using the two-stage stochastic model and the aforementioned distributed methodologies to solve FAM. A 10 - 15% increase in profitability is observed, and the optimality gaps from all the algorithms are less than 12%.

Finally, in the last chapter of this dissertation, we develop a search engine of basketball plays that can quickly find and prioritize similar plays from a large number of games (e.g., one season

or multiple seasons). With our system, coaches and scouts no longer need to spend dozens of hours watching video footages in order to find a play of interest. Given an input query, the system is able to return a ranked list of similar plays ordered by the relevance to the query in 5-7 seconds. Unlike traditional search engines (e.g., Google, Yahoo) which use keyword-based queries, we allow users to issue an input query either by selecting a play from previous games or by drawing a play of interest similar to how coaches draw up plays. We propose an encoded representation of basketball plays based on K-means clustering and player's role alignment to reduce the amount of data to be stored and facilitate fast identification of candidate plays. A key-value database is constructed to store the encoded plays and a streaming-based distributed framework is developed to compare candidate plays to the query in massive scale to achieve real time performance.

Personalization ranking is achieved via a pairwise learning to rank approach and the embedded similarity metric learnt through a convolutional neural network (CNN)-based autoencoder. Through mining the users' clickthrough logs and applying the rank function, our system is able to adapt to users' specific interest. In the end, we showcase the system's performance and the efficacy of our learning to rank approach by demonstrating rank quality in a user study.

The rest of the dissertation is organized as follows. Chapter 2 presents the model and solution approaches of the power grid visualization study. Chapter 3 solves the stochastic fleeting problem with MapReduce. Finally, in Chapter 4, we debut our large-scale information retrieval system of sports plays.

CHAPTER 2

# Power Grid Visualization by Means of Optimization

## 2.1. Motivation and Literature Review

Reliability of electric power transmission plays a critical role in almost every aspect of a modern society. Despite the advent of the smart grid and extensive automation associated with it, the reliability of the electric grid is still largely in the hands of human operators who manage it at control centers. Therefore, an effective way to prevent possible disruptions to the supply of electric power is to design a better display to allow human operators to process vast amount of information rapidly and reliably. A major difficulty associated with the design is the clutter from displaying too much information in too small of a display space. And it is further aggravated by the future need to broaden the views to also encompass neighboring control areas and display predicted values. Thus, optimizing the layout of visual elements consisting of substations and transmission lines to make the display interface less cluttered subject to human perceptual and cognitive capabilities is of vital significance.

To reduce clutter, geographically close substations should be spread out and transmission lines between them drawn, for example in a rectilinear pattern, to minimize line bends and intersections. Such patterns optimized with respect to given constraints can be modeled as a multicommodity flow problem with side constraints and a complex objective function over an embedded display grid. Unfortunately, due to the size of such a model, it cannot be solved

to a satisfactory quality within a reasonable computational time by any commercial software. Hence, developing effective solution methodologies is the focus of this research.

A stream of research that is closely related to ours is VLSI (very large scale integration) global routing. As one of the most challenging discrete optimization problems, plenty of research effort has been devoted into this field in past decades and among them, sequential routing, multicommodity flow-based methods and hierarchical methods are the most well-researched solution approaches. Chiang et al. [19] solve the problem by constructing Steiner min-max trees for each net sequentially. Shragowitz and Keel [64] were among the first researchers to work on global routing using a multicommodity flow model. The hierarchical method, first proposed by Burstein and Pelavin [13], is a systematic divide-and-conquer approach to transform the large and complicate overall problem into a series of subproblems. This menthod has been used by [54], [49], [33] and [34]. Hu and Sapatnekar [35] provide an extensive review of various VLSI global routing models and algorithms.

This chapter relies mostly on the hierarchical methods. In particular, we develop two network partitioning-based algorithms - a Lagrangian relaxation algorithm and a progressive hedging algorithm - to decompose the global network into smaller regions and then iteratively solve the subproblem in each region by readjusting penalties for corresponding solutions. They are like the traditional hierarchical methods at first glance, but we also incorporate several novel aspects.

(1) We apply clustering techniques such as $k$-means to capture the geographic closeness and connectivity between substations.

(2) Unlike VLSI global routing, the visualization decluttering problem addressed in this chapter also considers setting apart closely located substations so that when the transmission lines among them are routed, extremely cluttered areas can be avoided. Combining this location problem with VLSI greatly expands our model's fields of application, but also poses new computational challenges.

(3) The proposed iterative algorithms are based on Lagrangian relaxation and progressive hedging, which has been rarely used for solving VLSI routing problems. Unlike common Lagrangian relaxation procedure that relaxes the most difficult constraints or [81] which relaxes crosstalk constraints[1], we first partition the network into smaller regions so that the subproblem in each region can be easily solved, and then relax the compatibility requirements among regions. The progressive hedging algorithm works in a similar way by partitioning the network and solving each region.

We also develop a sequential routing approach to route transmission lines one by one based on a revised shortest path algorithm, and conduct extensive computational experiments to evaluate and compare the three proposed solution methodologies. Our findings show that the two network partitioning-based iterative algorithms deliver similar high quality solutions within 5 to 10 hours, while the sequential routing algorithm gives a preliminary result of much lower quality in a much shorter time. We also find out that the output from the iterative algorithms can be implemented immediately without modification, but the solution generated by the sequential approach possesses serious defects and needs further post-processing before being implemented. Thus, the comparison between the iterative and sequential methods is indeed a trade-off between computational resources and the quality of the solution.

[1]In electronics, the signal transmitted on one circuit or channel can create an undesired effect such as coupling in another circuit or channel. This phenomenon is called crosstalk.

The contribution of this chapter goes beyond the three methodologies to solve a challenging display visualization problem. Most importantly, we develop a divide-and-conquer approach based on mathematical programming techniques to solve large-scale network optimization problems, and demonstrate its applicability by performing a case study on a real-world power grid in the U.S. With slight adjustments, our work can be applied to VLSI and transportation networks. We also extend the traditional shortest path algorithm to incorporate several unique features such as line bends, crossings and overlaps, which have a spectrum of applications in transportation and logistics system design.

The rest of the chapter is organized as follows. Section 2.2 presents the model for the decluttering algorithm. Section 2.3 discusses the revised shortest path algorithm and the sequential routing approach. In Section 2.4, we formalize the construction and partitioning of the grid network and then present two network partitioning-based iterative algorithms. Section 2.5 gives the findings from computational experiments and Section 2.6 concludes.

## 2.2. Mathematical Formulation of the Problem

In order to minimize the clutter on a transmission control room display, a map of transmission lines is drawn in which the latitude and longitude coordinates of the substations are redefined and the transmission lines are rerouted on the lattice so that the following criteria are met.

(1) The displayed transmission lines should be piece-wise linear regardless of their true paths on the terrain so that (*a*) the total number of turns along the lines and (*b*) the total number of intersections between them are minimized.

(2) The overlaps between different substations and transmission lines must be minimized. In other words, all display elements must be as visible as possible to the operators regardless of the display scale.

(3) The total deviation from substations' actual geographic locations to the schematic coordinates must be minimized.

(4) The relative position between each pair of substations should be preserved as much as possible. That being said, given a substation and another substation south west of it, the algorithm should try to relocate them to points $A$ and $B$ so that $B$ is still south west of $A$ on the new map.

(5) The total number of loops should be minimized. We define a *loop* as a transmission line whose length is more than three times longer than the $L_1$ distance of its substations. An example of a loop is shown in Figure 2.1, in which the transmission line from $s$ to $t$ is a loop but the one from $m$ to $n$ is not.

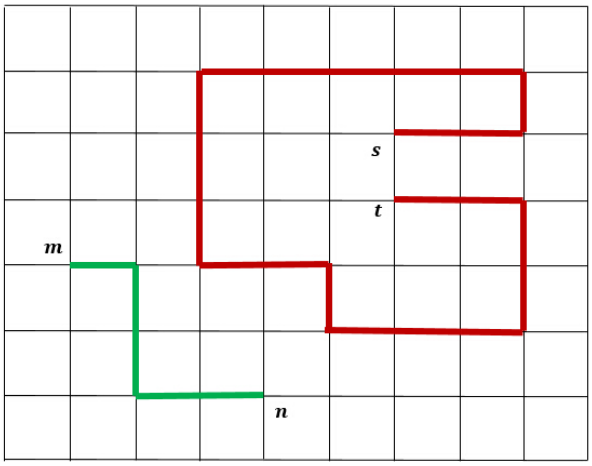


Figure 2.1. A graphical illustration of loops

The entire concept relies on the notion of an embedded orthogonal grid in which the edges consist of only vertical and horizontal lines. The intersections of edges are referred to as nodes, and they correspond to possible locations of substations. Although it suffices to have undirected edges for the sake of the display grid, modeling the layout requires directed arcs. The input to our model consists of a list of substations with latitude and longitude coordinates, and a list of pairs of substations that must be connected by a transmission line.

The following notation is used in the model.

**Sets**:

- $G = (N, A)$: the underlying network, in which $N$ is the set of all nodes (all possible locations of substations) and $A$ is the set of all arcs connecting the nodes.
- $E$: set of all underlying undirected edges. For each $e \in E$, let $e = \{a_1, a_2\}$ denote the two arcs in opposite directions defining $e$. We assume that each edge and arc is one of the two types. For simplicity, we call one type horizontal and the other type vertical.
- $S$: set of all substations.
- $M$: multi-set of all transmission lines. Each $m \in M$ is from $S \times S$. This is a multi-set because a pair of substations can have multiple lines. Pairs in $M$ are treated as ordered.
- $I_1(k)$, $O_1(k)$: set of all incoming, outgoing horizontal arcs of node $k$.
- $I_2(k)$, $O_2(k)$: set of all incoming, outgoing vertical arcs of node $k$.

**Decision Variables**:

$$x_{s,k} = \begin{cases} 1 \text{ if substation } s \in S \text{ is placed at node } k \in N \\ 0 \text{ otherwise} \end{cases}$$

$$y_{m,a} = \begin{cases} 1 \text{ if arc } a \in A \text{ is used to display transmission line } m \in M \\ 0 \text{ otherwise} \end{cases}$$

$$u_{m,k} = \begin{cases} 1 \text{ if transmission line } m \in M \text{ has a turn at node } k \in N \\ 0 \text{ otherwise} \end{cases}$$

$$v_{k,m_1,m_2} = \begin{cases} 1 \text{ if transmission lines } m_1,\ m_2 \in M \text{ have an intersection at node } k \in N \\ 0 \text{ otherwise} \end{cases}$$

$$o_e = \begin{cases} 1 \text{ if edge } e \in E \text{ is used by more than one transmission line} \\ 0 \text{ otherwise} \end{cases}$$

In line with criteria (1) - (5), the objective function of our decluttering model reads as:

$$\begin{aligned} \min\ & \alpha \sum_{m\in M}\sum_{k\in N} u_{m,k} + \beta \sum_{k\in N}\sum_{m_1\in M}\sum_{m_2\in M} v_{k,m_1,m_2} + \gamma \sum_{e\in E} o_e \\ & + \theta \sum_{s\in S}\sum_{k\in N} x_{s,k} d(s,k) \\ & + \mu \sum_{s_1\in S}\sum_{s_2\in S}\sum_{k_1\in N}\sum_{k_2\in N} 1_{\{x_{s_1,k_1}=x_{s_2,k_2}=1\}} V(s_1,s_2,k_1,k_2) \\ & + \nu \sum_{m\in M} 1_{\{\text{line } m \text{ is a loop}\}} \end{aligned} \tag{2.1}$$

where

- $\alpha, \beta, \gamma, \theta, \mu, \nu$ are penalty terms for turns, intersections, overlapping edges, deviations from actual geographic locations, violations of relative positions, and loops, respectively;

- $d(s,k)$ represents the distance from node $k$ to the actual geographic location of substation $s$;
- $1_{\{x_{s_1,k_1}=x_{s_2,k_2}=1\}}$ is an indicator variable indicating whether substations $s_1$ and $s_2$ are located at nodes $k_1$ and $k_2$, and $V(s_1,s_2,k_1,k_2) \in \{0,1\}$ indicates whether locating substations $s_1$ and $s_2$ at nodes $k_1$ and $k_2$ violates their relative position.

Notice that the 6$^{\text{th}}$ component of the objective function, $\nu \sum_{m\in M} 1_{\{\text{line } m \text{ is a loop}\}}$, is linked to decision variables in a complex way. For this reason, we approximate $\sum_{m\in M} 1_{\{\text{line } m \text{ is a loop}\}}$ by $\sum_{m\in M}\sum_{a\in A} y_{m,a}\text{length}(a)$ to minimize the total length of all the transmission lines. Although this approximation does not necessarily give the minimal number of loops, these two are (strongly) positively correlated.

The following constraints are imposed.

$$\sum_{k\in N} x_{s,k} = 1 \qquad s \in S \tag{2.2}$$

$$\sum_{s\in S} x_{s,k} \leq 1 \qquad k \in N \tag{2.3}$$

$$y_{m,a_1} + y_{m,a_2} \leq 1 \qquad e = \{a_1,a_2\} \in E,\ m \in M \tag{2.4}$$

$$x_{s,k} + \sum_{a\in I_1(k)\cup I_2(k)} y_{m,a} = \sum_{a\in O_1(k)\cup O_2(k)} y_{m,a} + x_{t,k} \qquad k \in N,\ m = (s,t) \in M \tag{2.5}$$

$$\sum_{a\in I_1(k)\cup I_2(k)} y_{m,a} \leq 1 \qquad k \in N, m \in M \tag{2.6}$$

$$\sum_{a \in O_1(k) \cup O_2(k)} y_{m,a} \leq 1 \qquad k \in N, m \in M \tag{2.7}$$

$$\sum_{m \in M} y_{m,a_1} + \sum_{m \in M} y_{m,a_2} - 1 \leq Lo_e \qquad e = \{a_1, a_2\} \in E \tag{2.8}$$

$$y_{m,r} + y_{m,h} - 1 \leq u_{m,k} \qquad k \in N,\ m \in M,\ r \in I_1(k), h \in O_2(k) \text{ or } r \in I_2(k), h \in O_1(k) \tag{2.9}$$

$$y_{m_1,r_1} + y_{m_1,h_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{k,m_1,m_2} \qquad k \in N,\ m_1, m_2 \in M,\ r_1 \in I_1(k), h_1 \in O_1(k),\ r_2 \in I_2(k), h_2 \in O_2(k) \text{ or } r_1 \in I_2(k), h_1 \in O_2(k), r_2 \in I_1(k), h_2 \in O_1(k) \tag{2.10}$$

$$x_{s,k},\ y_{m,a},\ u_{m,k},\ v_{k,m_1,m_2},\ o_e \in \{0,1\} \tag{2.11}$$

Constraints (2.2) assign each substation to a node. Constraints (2.3) and (2.4) impose that we can assign at most one substation to a node and no two-arc cycles are allowed in the network. Constraints (2.5) - (2.7) model the transmission lines in the network using the following common principles.

(1) There is a single arc emanating from the origin of a transmission line;

(2) There is a single arc going into the destination node of a transmission line;

(3) For all other nodes, the *flow-in* must equal to the *flow-out*.

Constraints (2.5) become standard flow conservation constraints if $x_{s,k} = x_{t,k} = 0$ for a given $k \in N$. If either is 1, (2.5) - (2.7) combined imply the corresponding node is either the origin

or destination. Finally, constraints (2.8) - (2.10) link the decision variables capturing the objectives, where $L = 2|M|$ (or any larger number). The model is NP-hard. However, there are some constrained versions of the problem that are proven to be NP-complete:

(1) If overlaps and intersections are not allowed and substations have a fixed position on the grid, the problem was independently shown to be NP-complete by [41] and [56].

(2) If substations can be arbitrarily relocated, and overlaps and intersections are admitted but turns are not admitted, then the problem is NP-complete according to [26].

(3) If overlaps, intersections and turns are not allowed, the problem of relocating the substations on a grid so that their horizontal and vertical ordering is maintained is again NP-complete by [53].

These constrained versions have wide applications in routing and display technology, but none of them is able to give a decluttering solution that satisfies criteria (1) - (5). Hence, in this chapter, we focus on objective function (2.1).

## 2.3. A Sequential Routing Algorithm

The oldest and perhaps the most straightforward approach to route multiple lines on a grid is to pick an order and then route them sequentially. In this section, we develop a sequential routing algorithm based on a revised version of the Dijkstra's shortest path algorithm.

### 2.3.1. Locating the Substations

As we can see from (2.1) - (2.11), we create far more arc-related variables (i.e., $y$'s and hence the associated $u$'s, $v$'s and $o$'s) than node-related variables (i.e., $x$'s). This implies that a hierarchical solution approach that first locates each substation to a node by solving a partial optimization

model with only node-related variables and then routes the transmission lines is likely to reduce the computational load significantly. The partial model is formulated as follows.

$$\min \quad \sum_{s\in S}\sum_{k\in N} x_{s,k}d(s,k) \tag{2.12}$$

subject to

$$\sum_{k\in N} x_{s,k} = 1 \qquad s \in S \tag{2.13}$$

$$\sum_{s\in S} x_{s,k} \leq 1 \qquad k \in N \tag{2.14}$$

$$x_{s_1,k_1} + x_{s_2,k_2} \leq 1 \qquad if\ V\left(s_1,k_1,s_2,k_2\right) = 1,\ s_1,s_2 \in S,\ k_1 \in neighbor(s_1),\ k_2 \in neighbor(s_2) \tag{2.15}$$

$$x_{s,k} \in \{0,1\} \tag{2.16}$$

We must also ensure that the substations do not deviate too much from their actual geographic coordinates. This can be easily done by setting $x_{s,k} = 0$ for all nodes $k$ with distance $d(s,k) \geq l$, where $l$ is a threshold.

Notice that instead of minimizing pairwise geographical violations in the objective function, a new set of constraints (2.15) are introduced so that for each pair of substations $s_1$ and $s_2$, any possible violation caused by relocating $s_1$ and $s_2$ within their neighborhoods is forbidden. The

neighborhood of substation $s$, $neighbor(s)$, is defined as the set of grid nodes within a certain distance $l'$ from $s$. Following this definition, contraints (2.15) prevent most of the violations because according to (2.12), substations would most likely be located within their corresponding neighborhoods. We make this modification to avoid millions of calculations in the objective function to facilitate the solution process, but since it is weaker in terms of forbidding pairwise geographical violations, we only use it in our sequential routing heuristic.

Since the partial model is much simpler than the overall problem, we can afford a relatively fine grid to put substations close to their original coordinates and avoid many pairwise geographical violations. A straightforward choice would be an $|S| \times |S|$ grid by drawing a pair of latitude and longitude lines at the coordinates of each substation. We can certainly create a finer grid, even an arbitrarily fine one to completely rule out intersections and overlaps, but meanwhile the routing process will also become computationally challenging. To maintain the balance, we stick with the $|S| \times |S|$ grid.

### 2.3.2. A Revised Dijkstra's Shortest Path Algorithm

Once locating each substation to a node on the grid, the next step is to route the transmission lines to capture the remaining components in the objective function: minimization of turns, intersections, overlapping edges, and the total length of all the transmission lines. To do so, we first introduce the following revised Dijkstra's shortest path algorithm, which performs exactly the same as the general Dijkstra's algorithm, except that each time when we consider all the unvisited neighbors of a visited node and update the tentative distances, we need to trace back to the second-to-last node along the current shortest path to check for turns and intersections.

This revised Dijkstra's algorithm exhibited in Algorithm 1 is used in several steps of the overall solution methodologies.

**Algorithm 1** Revised Dijkstra's shortest path algorithm

```
1:  Initialize dist[s] := 0, prev[s] := ∅
2:  for each v ∈ V − {s} do
3:      dist[v] := ∞
4:      prev[s] := ∅
5:  end for
6:  Set S := ∅, Q := V
7:  while Q ≠ ∅ do
8:      Select u ∈ Q with the minimum dist[u]
9:      Set S := S ∪ {u}
10:     Set z := prev[u]
11:     for each neighbor v of u do
12:         if arcs (u,v) and (z,u) are perpendicular then
13:             if dist[v] > dist[u] + w(u,v) + α then
14:                 dist[v] := dist[u] + w(u,v) + α
15:                 prev[v] := u
16:             end if
17:         else
18:             if there exists another already processed line going through u and perpendicular to
                (z,v) then
19:                 if dist[v] > dist[u] + w(u,v) + β then
20:                     dist[v] := dist[u] + w(u,v) + β
21:                     prev[v] := u
22:                 end if
23:             else
24:                 if dis[v] > dist[u] + w(u,v) then
25:                     dist[v] := dist[u] + w(u,v)
26:                     prev[v] := u
27:                 end if
28:             end if
29:         end if
30:     end for
31:     Remove u from Q
32: end while
33: return dist, prev
```

In this algorithm, $s$ is the source node, $V$ is the set of all nodes, $w(u,v)$ denotes the weighted length of arc $(u,v)$ and prev$[u]$ represents the previous node of $u$ along the current best path from source to $u$. Turns and intersections are penalized with $\alpha$ and $\beta$ penalties and we can proportionately penalize the total length of transmission lines by assigning proper weight (e.g., $v$) to arcs. Since we only add two steps - checking the occurrence of turns in Steps 11 - 15 and intersections in Steps 17 - 21 and adding penalties accordingly - in each iteration of the original Dijkstra's algorithm, the complexity remains the same.

We should also be aware that this revised Dijkstra's algorithm does not guarantee an optimal solution. An example in which the algorithm fails to find an optimal solution is given in Figure 2.2. To find the revised shortest path from $A$ to $D$ with the minimum length and penalty, Algorithm 1 follows the following sequence: (1) Starting from node $A$, we update dist$[B] = 2$ and dist$[B'] = 2$. (2) Breaking the tie by randomly choosing node $B$, we update dist$[C] = 5$, dist$[G] = 6$ and dist$[E] = 26$. (3) At node $B'$, we update dist$[F] = 7$ but do not update dist$[C]$ as we cannot get a better path traversing from $B'$ to $C$. So the best path from source node to $C$ at this moment is still $A \to B \to C$. (4) Finally, at node $C$, we update dist$[D] = 10$ and dist$[H] = 10$. Since dist$[D]$ cannot be further updated from $E$ or other neighborhood nodes, we get our solution as $A \to B \to C \to D$ with total length and penalty equal to 10. However, a better path $A \to B' \to C \to D$ with length and penalty equal to 9 could be obtained if at node $C$, we can trace back to node $B'$ rather than $B$ and hence avoid the turn penalty incurred along $B \to C \to D$. Unfortunately, Algorithm 1 cannot achieve this because node $B'$ is not on the best path to $C$ when updating dist$[C]$.

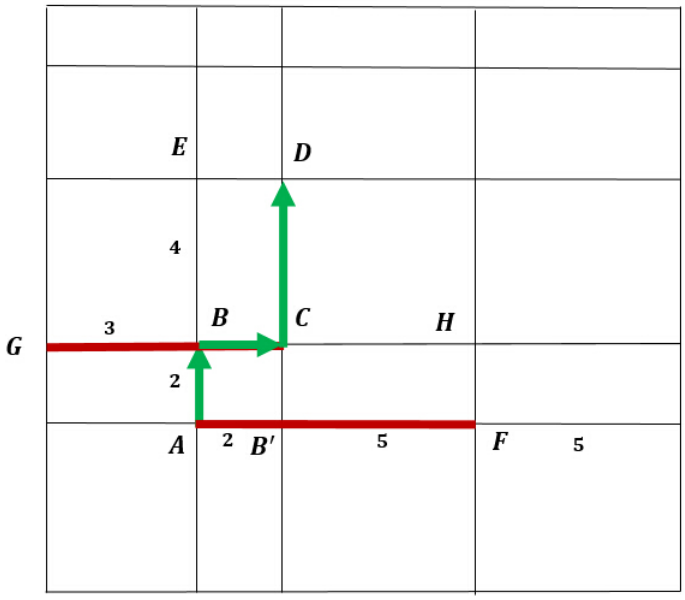


Figure 2.2. A counter-example of the revised Dijkstra's algorithm. The penalty terms are assigned as $\alpha = 1$, $\beta = 20$ and weights of edges/arcs are as labeled. The bold lines $CG$ and $AF$ refer to the fact that prior lines have already been routed on these edges. The bold line with arrow $A \to B \to C \to D$ denotes the solution obtained from Algorithm 1.

### 2.3.3. The Sequential Routing Algorithm

Before applying Algorithm 1 to the relocated substations, we need to assign weights to the arcs. For each arc $a \in A$, we assign $w(a) := \nu \times length(a)$, which is proportional to its length and effectively imposes a penalty of $\nu$. If the underlying undirected edge has already been used by other transmission lines, $w(a)$ is updated to $w(a) := w(a) + \gamma$ to capture the penalty cost for overlaps. We formalize the overall sequential routing algorithm as follows.

The primary drawback of this sequential approach is that the quality of the solution highly depends on the order in which the transmission lines are processed, and there is no systematic way of finding a good order. Furthermore, routing of transmission lines and relocating of substations are determined separately with respect to partial objectives, and by doing so, the

**Algorithm 2** The sequential routing algorithm

```
Construct the |S| × |S| grid
Locate each substation to a grid node by solving (2.12) - (2.16)
Sort to-be-connected transmission lines into descending order with respect to the L1 distance between terminal substations; let Ms denote the set of sorted lines
for k ∈ Ms do
    Call Algorithm 1 to generate path Pk
    for each arc a ∈ Pk do
        if the underlying undirected edge e has not been used by other lines than Pk then
            Set w(a) := w(a) + γ
        end if
    end for
end for
```

correlation between these two families of objectives are neglected. Later in Sections 2.4.2 and 2.4.3, we develop solution approaches to consider them together.

Despite these drawbacks, since we do not solve the overall problem directly, the running time should be low. This method also allows the cluttering information (e.g., turns, intersections and overlaps) for previously routed lines to be explicitly considered when routing a new line, thus providing a quality feasible solution to the problem formulated in (2.1) - (2.11).

## 2.4. Two Network Partitioning-Based Optimization Algorithms

We start this section by constructing an appropriately-sized grid and introducing a partitioning algorithm with which the constructed grid network can be decomposed into smaller solvable regions. We then present two mathematical programming-based algorithms - one Lagrangian relaxation algorithm and one progressive hedging algorithm - to iteratively solve the problem in each region. Although the two algorithms are based on the same (partitioned) network, they rely on completely different assumptions to handle the boundaries.

### 2.4.1. Network Construction and Partitioning

Unlike in the sequential routing approach, since all the objectives and transmission lines are considered concurrently in this section, an $|S| \times |S|$ grid would yield intractable subproblems despite the partitioning and is thus unacceptable. At the same time, however, we still need to generate enough grid nodes and arcs to place substations and route transmission lines. Therefore, the algorithm to be exhibited next is developed to find the areas with high substation densities and to create more grid lines within these areas. We start with a sparse grid and the algorithm adjusts the discretization granularity along the way. The next step of the algorithm is to group grid nodes into regions so that:

- Geographically close substations are more likely to be located in the same region;
- Substations with a transmission line connecting between them are more likely to be located in the same region, or equivalently, the algorithm should discourage inter-region transmission lines;
- The number of inter-region arcs should be as small as possible.

We propose the following construction and partitioning approach, with $n, n_1, n_2$ and $n_3$ being parameters to control the granularity of the grid discretization.

(1) **Group substations into regions and generate the initial grid network.**

A simple distance function using $L_2$ distance between substations and *k-means* clustering algorithm are applied to divide the substations into $n_1$ regions. Within each region, we generate latitude lines according to the latitudes of the north- and south-most substations. And similarly, we generate longitude lines according to the longitudes of the

east- and west-most substations. This gives us the initial network grid with $4n_1$ lines (if none of them overlap).

(2) **Find areas with the highest substation densities and create a finer granularity grid within these areas.**

Given the initial network grid, the number of substations in each grid rectangle excluding substations falling on grid edges is calculated. Extra latitude and longitude lines are created in the $n_2$ grid rectangles with the highest number of substations to divide each of them into 4 equal-sized rectangles. We repeatedly perform this until no grid rectangle contains more than $n_3$ substations, and convert the final grid network into a directed graph.

(3) **Define a distance function for grid nodes.**

In order to incorporate the connectivity of substations into the partitioning algorithm, we need the following definition of periphery.

**Definition 2.4.1.** *For any substation $s \in S$, the periphery of $s$ is defined by the following three rules:*

a. *If substation $s$ falls on the intersection of grid arcs, the periphery of $s$ is defined as the eight nodes surrounding $s$ plus the node on which $s$ falls to.*

b. *If substation $s$ falls on a single grid arc, the periphery of $s$ is defined as the six nodes surrounding $s$.*

c. *If substation $s$ does not fall on any grid arc, the periphery of $s$ is defined as the four nodes surrounding $s$.*

A graphical illustration of these three rules is given in Figure 2.3. Before performing clustering on grid nodes, we define a new distance function to capture both the closeness and the connectivity of substations: given any two nodes $i$ and $j$, the distance between $i$ and $j$ is defined as half of the $L_2$ distance between them if there exist substations $s$ and $t$ connected by a transmission line such that $i$ is in the periphery of $s$ and $j$ is in the periphery of of $t$; otherwise, the distance between $i$ and $j$ is simply the $L_2$ distance between them. Halving $L_2$ distance effectively brings potential locations of connected substations closer.

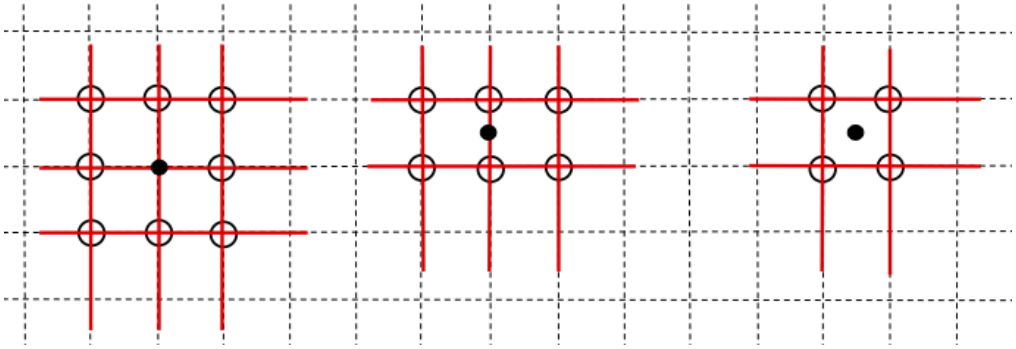

Figure 2.3. A graphical illustration of periphery: the bold dot represents the geographic location of a substation and the circles represent its periphery nodes

4). **Group grid nodes into regions.**

Given the distances between grid nodes, *k-means* clustering algorithm is applied to divide the nodes into $n$ regions.

5). **Reduce the number of inter-region arcs.**

A tabu search algorithm attempting to assign boundary nodes to an adjacent region to reduce the number of inter-region arcs is developed. To learn more about tabu search and its applications, we refer readers to [28] and [29].

The above approach produces contiguous regions and an unevenly-spaced grid network.

### 2.4.2. The Lagrangian Relaxation Algorithm

Lagrangian relaxation is a common approach for solving difficult problems. The basic principle is that the relaxed problem should be solved very easily for fixed values of Lagrange multipliers. Since applying the algorithm developed in Section 2.4.1 partitions the network and the subproblem corresponding to each region is of much smaller size, we relax constraints pertaining to the boundaries and iteratively solve the relaxed model in each region. We start by introducing the following region specific notation and adjusting our model to the partitioned grid.

- Let $S_i$ be the set of substations located in region $i$, $i = 1,...,n,\ S = S_1 \cup \cdots \cup S_n$.
- Let $N_i$ be the set of interior nodes in region $i$ and $N_i'$ be the set of boundary nodes in region $i$. The interior nodes are connected only by interior arcs, and the boundary nodes connect to at least one inter-region arc.
- Let $A_i$ be the set of interior arcs in region $i$ and $A_{i,j}$ be the set of inter-region arcs from region $i$ to $j$, where $j \in adj(i)$ and $adj(i)$ is the set of regions sharing at least one inter-region arc with region $i$. Let $\bar{A}_i = A_i \cup \bigcup_{j \in adj(i)} A_{i,j} \cup \bigcup_{j \in adj(i)} A_{j,i}$ denote all the arcs related to region $i$.
- Let $E_i$ be the set of undirected interior edges in region $i$ and let $\bar{E}_i$ denote all the edges related to region $i$, $\bar{E}_i = E_i \cup \{\text{the set of inter-region edges incident to nodes of } N_i'\}$.
- Let $M_i$ be the set of interior transmission lines in region $i$, whose connected substations are both located in $i$. Let $M_{i,j}$ be the set of inter-region transmission lines from region $i$ to an adjacent region $j$. Similarly, let $\bar{M}_i = M_i \cup \bigcup_{j \in adj(i)} M_{i,j} \cup \bigcup_{j \in adj(i)} M_{j,i}$ denote all the lines related to region $i$.
- Let $I_1^i(k), O_1^i(k)$ be the set of incoming, outgoing horizontal arcs of node $k$ within region $i$.

· Let $I_2^i(k), O_2^i(k)$ be the set of incoming, outgoing vertical arcs of node $k$ within region $i$.

The decision variables remain the same, and the objective function only needs a slight modification to reflect the partitioning.

$$
\begin{aligned}
\min \quad & \alpha \sum_{i}^{n} \sum_{m \in \bar{M}_i} \sum_{k \in N_i \cup N_i'} u_{m,k} + \beta \sum_{i}^{n} \sum_{k \in N_i \cup N_i'} \sum_{m_1 \in \bar{M}_i} \sum_{m_2 \in \bar{M}_i} v_{k,m_1,m_2} \\
& + \gamma \sum_{e \in E} o_e + \theta \sum_{i}^{n} \sum_{s \in S_i} \sum_{k \in N_i \cup N_i'} x_{s,k} d(s,k) \\
& + \mu \sum_{i}^{n} \sum_{s_1 \in S_i} \sum_{s_2 \in S_i} \sum_{k_1 \in N_i \cup N_i'} \sum_{k_2 \in N_i \cup N_i'} 1_{\{x_{s_1,k_2} = x_{s_2,k_2} = 1\}} V(s_1,k_1,s_2,k_2) \\
& + \mu \sum_{i}^{n} \sum_{s_1 \in S_i} \sum_{s_2 \in S \setminus S_i} \sum_{k_1 \in N_i \cup N_i'} x_{s_1,k_1} V'(s_1,k_1,s_2) \\
& + \nu \sum_{m \in M} \sum_{a \in A} y_{m,a} \text{length}(a)
\end{aligned}
\tag{2.17}
$$

where, $V'(s_1,k_1,s_2)$ is an indicator of whether locating substation $s_1$ at node $k_1$ violates the relative position between $s_1$ and $s_2$.

In (2.17), we see that the $5^{th}$ term of the original objective function has been replaced by two components to indicate: (1) whether locating substations $s_1$ and $s_2$ at nodes $k_1$ and $k_2$ in the same region $i$ violates their relative positions; and (2) whether locating substation $s_1$ at node $k_1$ in region $i$ violates the relative positions between $s_1$ and the other substations outside $i$.

To facilitate the relaxation of boundary constraints, we need to model the interior network and the boundaries separately. Constraints (2.2) - (2.4) and (2.6) - (2.8) can be easily adapted

within each region, and to model the flow of transmission lines, especially at boundaries, we split (2.5) into two cases.

(1) Interior transmission lines whose connected substations locate in the same region $i = 1,2,...,n$ are modeled in the same way as (2.5).

$$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} + x_{t,k} \qquad k \in N_i \cup N_i',\ m = (s,t) \in M_i \tag{2.18}$$

(2) Let $c = (p,q) \in A_{i,j}$ be an inter-region arc from region $i$ to $j$ and $c' = (q,p) \in A_{j,i}$ be the corresponding reverse arc from region $j$ to $i$, where $p \in N_i'$ and $q \in N_j'$. To handle an inter-region transmission line whose connected substations locate in two adjacent regions $i$ and $j$, we should first route the two ends separately in $i$ and $j$ to some boundary nodes $p$ and $q$ and then connect the two segments through the inter-region arc across the boundary.

$$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j} \tag{2.19}$$

$$x_{t,k} + \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} = \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i} \tag{2.20}$$

$$x_{s,p} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} = y_{m,c} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} \quad m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j} \tag{2.21}$$

$$x_{t,p} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} = y_{m,c'} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} \quad m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i} \tag{2.22}$$

$$y_{m,c} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} \leq 1 \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j} \tag{2.23}$$

$$y_{m,c'} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} \leq 1 \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i} \tag{2.24}$$

The routing is modeled by constraints (2.19) and (2.20), and the connection is accomplished through constraints (2.21) and (2.22).

Finally, we have constraints capturing turns and intersections. To model turns and intersections which do not involve inter-region arcs, the constraints remain the same as (2.9) and (2.10). On the other hand, modeling turns and intersections constructed by inter-region arcs would need a substitution of an outgoing arc (i.e., $h$ in (2.9) and $h_1$ in (2.10)) by the corresponding inter-region arc.

$$y_{m,c} + y_{m,r} - 1 \leq u_{m,p} \qquad m \in \bigcup_{j \in adj(i)} M_{i,j},\ r \in I_1^i(p) \text{ or } I_2^i(p) \tag{2.25}$$

$$y_{m,c'} + y_{m,r} - 1 \leq u_{m,p} \qquad m \in \bigcup_{j \in adj(i)} M_{j,i},\ r \in O_1^i(p) \text{ or } O_2^i(p) \tag{2.26}$$

(2.27)

$$y_{m_1,c} + y_{m_1,r_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{p,m_1,m_2} \qquad m_1 \in \bigcup_{j \in adj(i)} M_{i,j},\ m_2 \in \bar{M}_i,$$

$$r_1 \in I_1^i(p),\ r_2 \in I_2^i(p),\ h_2 \in O_2^i(p) \text{ or}$$

$$r_1 \in I_2^i(p),\ r_2 \in I_1^i(p),\ h_2 \in O_1^i(p)$$

(2.28)

$$y_{m_1,c'} + y_1 m,r_1 + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{p,m_1,m_2} \qquad m_1 \in \bigcup_{j \in adj(i)} M_{j,i},\ m_2 \in \bar{M}_i,$$

$$r_1 \in O_1^i(p),\ r_2 \in I_2^i(p),\ h_2 \in O_2^i(p) \text{ or}$$

$$r_1 \in O_2^i(p),\ r_2 \in I_1^i(p),\ h_2 \in O_1^i(p)$$

The complete formulation ready for Lagrangian relaxation is presented in the Appendices. Notice that this formulation is unable to capture the transmission lines going across more than two regions, and hence is not equivalent to the original model. We call such transmission lines as *jump lines* to indicate that they can *jump* over one or several regions. Fortunately, in our New York State case, the jump lines only account for less than 5% of all the transmission lines, so they can be taken care of by a sequential routing heuristic after most others are processed. But if this percentage is too high[2], post-processing jump lines may not be a viable option.

With this formulation, all the constraints related to the inter-region arcs are relaxed. For each inter-region edge $e = \{c, c'\}$, $\lambda_e$ is the Lagrange multiplier for constraint (2.8) and $\lambda_{m,c}^{1(i)}$, $\lambda_{m,c'}^{1(i)}$, $\lambda_{m,c}^{2(i)}$, $\lambda_{m,c'}^{2(i)}$ are for constraints (2.21) - (2.24). Moreover, multipliers $\lambda_{m,c}^{3(i)}$, $\lambda_{m,c}^{4(i)}$, $\lambda_{m,c'}^{3(i)}$, $\lambda_{m,c'}^{4(i)}$, $\lambda_{m,c}^{5(i)}$, $\lambda_{m,c}^{6(i)}$ $\lambda_{m,c'}^{5(i)}$, $\lambda_{m,c'}^{6(i)}$ are for constraints (2.25) - (2.28) since each of them consists of two different cases.[3] An iterative algorithm is then developed to solve the Lagrangian relaxation, in which we improve the handling of inter-region transmission lines through the notion of penalties that encourage these lines to be more compatible at boundaries in each iteration. Having the grid nodes grouped into $n$ regions, the Lagrangian relaxation algorithm is described as follows.

[2]This is unusual because long transmission lines would cause severe loss of energy and are thus discouraged in the field.

[3]If edge $e$ falls on the borderline of the entire grid network, each of the constraints (2.25) - (2.28) consists of only one case.

(1) **Group substations into *n* regions.**

Notice that the partitioning algorithm is performed on the grid network (i.e., nodes) rather than on substations. Therefore, in order to assign each substation into a region, we have to establish a correspondence between substations and nodes. We use (2.12) - (2.16) from Section 2.3.1 to initially locate each substation $s$ to a node $k$. Then $s$ can be assigned to the region which $k$ belongs to.

(2) **Solve the relaxed problem.**

Since all the constraints pertaining to the inter-region arcs are relaxed, the problem decomposes into $n$ smaller subproblems. They are well-defined and can be solved independently. For fixed values of Lagrange multipliers, the values at the boundaries (i.e., solution for $y_{m,c}$) are optimally determined by the following rule:

$$y_{m,c} = \begin{cases} 0 & \text{if } \lambda_e - (\lambda_{m,c}^{1(i)} + \lambda_{m,c}^{1(j)}) + \left(\sum_{k=2}^{6} \lambda_{m,c}^{k(i)} + \sum_{k=2}^{6} \lambda_{m,c}^{k(j)}\right) + \nu \text{ length}(c) > 0 \\ 1 & \text{otherwise} \end{cases}.$$

Intuitively, we are assigning *easy-going* arcs to the boundaries because the above *if*-statement sums up all the Lagrange multipliers associated with $c$ and hence effectively indicates how difficult it is to go across this inter-region arc for any transmission line. We then compute the gradients within each region. For example, in region $i$, the gradient of constraint (2.25) is computed as $y_{m,c} + y_{m,r} - 1 - u_{m,p}$ after the subproblem is solved.

(3) **Find a primal feasible solution based on a Lagrangian solution.**

The paths obtained from solving the relaxed subproblems are unlikely to be feasible

for the original problem. However, we can readily find a feasible solution from these paths by the following heuristic.

a. For any transmission line $m = (s,t)$ with both substations $s$ and $t$ in the same region, we locate $s$, $t$ and route $m$ according to the solution from the subproblem.

b. For any transmission line $m = (s,t)$ with substations $s$ and $t$ in different regions, we first locate $s$, $t$ according to the solution from the subproblem and then find the shortest path between $s$ and $t$ using Algorithm 1. The weight of interior arc is set as $\nu$ times its length, and to avoid routing the line across boundaries frequently, the weight of any inter-region arc $a$ is assigned as $w(a) := L' \times \nu \times \text{length}(a)$. Here $L'$ is an amplifying parameter.

The order of processing or routing the lines is determined by $\sum_c \left[ \left( \sum_{k=2}^{6} \lambda_{m,c}^{k(i)} + \sum_{k=2}^{6} \lambda_{m,c}^{k(j)} \right) - \left( \lambda_{m,c}^{1(i)} + \lambda_{m,c}^{1(j)} \right) \right]$, which indicates the level of *difficulty* to pass through the boundaries for each transmission line $m \in M$. $c$ denotes any inter-region arc. To be specific, if

$$\sum_c \left[ \left( \sum_{k=2}^{6} \lambda_{m_1,c}^{k(i)} + \sum_{k=2}^{6} \lambda_{m_1,c}^{k(j)} \right) - (\lambda_{m_1,c}^{1(i)} + \lambda_{m_1,c}^{1(j)}) \right] > \sum_c \left[ \left( \sum_{k=2}^{6} \lambda_{m_2,c}^{k(i)} + \sum_{k=2}^{6} \lambda_{m_2,c}^{k(j)} \right) - (\lambda_{m_2,c}^{1(i)} + \lambda_{m_2,c}^{1(j)}) \right],$$

$m_1$ will be routed before $m_2$ in the above heuristic.

(4) **Update the Lagrange multipliers.**

Stepsize $t^{(n)} = a_0 \times a_1^n$ at the $n^{th}$ iteration is used, where $a_0$ and $a_1$ are randomly selected between 0 and 2. The Lagrange multipliers are then updated by $\lambda^{(n+1)} = \lambda^{(n)} + t^{(n)}(\text{gradient})$.

The following Algorithm 3 summarizes the overall procedure.

**Algorithm 3** Lagrangian relaxation algorithm

1: Construct and partition the network grid into $n$ regions
2: Assign substations into each region by solving (2.12) - (2.16)
3: Temporarily remove all the jump lines
4: Initialize Lagrange multipliers, $LB := -\infty$, $UB := \infty$, and $a_0$, $a_1$
5: Set $k := 1$
6: **for** $i = 1, ..., n$ **do**
7: Solve the relaxed subproblem in region $i$
8: Compute the gradients related to the subproblem in region $i$
9: **end for**
10: Update $LB$
11: Construct a feasible solution, and update $UB$
12: Set $t^{(k)} := a_0 \times a_1^k$
13: **if** stopping criteria are not met (e.g., $UB - LB > \varepsilon$) **then**
14: Update Lagrange multipliers: $\lambda^{(k+1)} := \lambda^{(k)} + t^{(k)}(\text{gradient})$
15: Set $k := k + 1$
16: Go to step 6
17: **end if**
18: Call Algorithm 1 to route jump lines sequentially

### 2.4.3. The Progressive Hedging Algorithm

Progressive hedging (PH) proposed by Rockafellar and Wets [57] is a decomposition type method for solving multi-stage stochastic programming problems. The basic idea is to iteratively solve individual scenario problems, perturbed in a certain sense, and to aggregate the scenario-dependent solutions into an overall implementable solution. Under certain assumptions, the sequence of the implementable solutions converges to the solution of the stochastic program. However, in the presence of discrete variables, PH is only a heuristic (see [24] and [45]).

In this section, we present a PH algorithm by considering the subproblem in each region as a scenario. Unlike the Lagrangian relaxation approach in which inter-region arcs do not belong

to any region, the PH approach partitions the network in such a way that each region includes inter-region arcs either originating from or going into it.

The notation and decision variables are defined in the same way as Section 2.4.2, except that within each region the interior nodes and boundary nodes are no longer differentiated. Thus, $N_i$ denotes all the nodes in region $i$. In addition, since each inter-region arc now belongs to both regions it connects to, we have two scenario-dependent solutions obtained from solving for the arc in different regions. In order to consolidate the scenario-dependent solutions into an implementable solution, we must force them to be equal.

The objective function is different from (2.17) since we have to take scenarios/regions into consideration. It reads as

$$\min \sum_{i}^{n} Q_1^i + 0.5 \sum_{i}^{n} Q_2^i \tag{2.29}$$

where

$$\begin{aligned}
Q_1^i = {} & \alpha \sum_{m \in \bar{M}_i} \sum_{k \in N_i} u_{m,k} + \beta \sum_{k \in N_i} \sum_{m_1 \in \bar{M}_i} \sum_{m_2 \in \bar{M}_i} v_{k,m_1,m_2} \\
& + \gamma \sum_{e \in E_i} o_e + \theta \sum_{s \in S_i} \sum_{k \in N_i} x_{s,k} d(s,k) \\
& + \mu \sum_{s_1 \in S_i} \sum_{s_2 \in S_i} \sum_{k_1 \in N_i} \sum_{k_2 \in N_i} 1_{\{x_{s_1,k_1} = x_{s_2,k_2} = 1\}} V(s_1,k_1,s_2,k_2) \\
& + \mu \sum_{s_1 \in S_i} \sum_{s_2 \in S \setminus S_i} \sum_{k_1 \in N_i} x_{s_1,k_1} V'(s_1,k_1,s_2) \\
& + \nu \sum_{m \in \bar{M}_i} \sum_{a \in A_i} y_{m,a} \text{length(a)}, \\
Q_2^i = {} & \gamma \sum_{e \in \bar{E}_i \setminus E_i} o_e + \nu \sum_{m \in \bar{M}_i \setminus M_i} \sum_{a \in \bar{A}_i \setminus A_i} y_{m,a} \text{length(a)}.
\end{aligned}$$

As we can see from (2.29), the summation is over all regions. In each region $i = 1,2,...,n$, $Q_1^i$ consists of the objective components that are independent of inter-region arcs, while $Q_2^i$ has objectives related to those arcs. Each inter-region arc connects exactly 2 regions, and thus the term 0.5 is in the objective function as a probability.

Constraints (2.2) - (2.4) and (2.6) - (2.10) are adapted for each region $i = 1,2,...,n$ in a straightforward way, and constraints (2.18) remain the same for modeling interior transmission lines. The following two constraints are unique to the PH approach.

$$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1(k) \cup O_2(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j} \tag{2.30}$$

$$x_{t,k} + \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} = \sum_{a \in I_1(k) \cup I_2(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i} \tag{2.31}$$

Together, they connect the two ends of an inter-region transmission line to some inter-region arcs at the boundary. For each pair of adjacent regions $i$ and $j$, we also need to impose the following constraints to make the scenario-dependent solutions implementable.

$$y_{m,a}^i = y_{m,a}^j \qquad j \in adj(i),\ a \in A_{i,j},\ m \in \bigcup_j M_{i,j}, \tag{2.32}$$

where $y_{m,a}^k$ is the value of $y_{m,a}$ from solving scenario $k$, $k = i, j$. Similar to Section 2.4.2, this new formulation cannot take jump lines into explicit consideration either.

Like the Lagrangian relaxation approach, we first partition the network into $n$ regions and assign each substation into a region by solving (2.12) - (2.16). Next, due to the model's intrinsic

inability of handling jump lines, these lines are temporarily removed before the start of the iterative algorithm and then processed after most other lines have been routed. We now formalize our PH algorithm as follows, taking $\rho$ (penalty factor) as an input parameter.

**Algorithm 4** Progressive hedging algorithm

1: Set $k := 0$
2: Initialize $\bar{y}^{(0)}_{m,a}$ and $(w^i_{m,a})^{(0)}$, $i = 1,2,...,n$ for each inter-region arc $a$ and each inter-region line $m$
3: Set $k := k+1$
4: **for** $i = 1,2,...,n$ **do**
5: Solve subproblem $i$ and obtain

$$(x^i, y^i, o^i, u^i, v^i)^{(k)} := \arg min \left( Q^i_1 + Q^i_2 + \sum_{m \in \bar{M}_i \setminus M_i} \sum_{a \in \bar{A}_i \setminus A_i} (w^i_{m,a})^{(k-1)} y_{m,a} + \frac{\rho}{2} \sum_{m \in \bar{M}_i \setminus M_i} \sum_{a \in \bar{A}_i \setminus A_i} \| y_{m,a} - \bar{y}^{(k-1)}_{m,a} \|^2 \right) \tag{2.33}$$

6: **end for**
7: **for** any pair of adjacent regions $i$ and $j$ **do**
8: **for** each inter-region arc $a$ and each inter-region line $m$ between $i$ and $j$ **do**
9: $\bar{y}^{(k)}_{m,a} = 0.5(y^i_{m,a})^{(k)} + 0.5(y^j_{m,a})^{(k)}$,
$(w^i_{m,a})^{(k)} = (w^i_{m,a})^{(k-1)} + \rho[(y^i_{m,a})^{(k)} - \bar{y}^{(k)}_{m,a}]$,
$(w^j_{m,a})^{(k)} = (w^j_{m,a})^{(k-1)} + \rho[(y^j_{m,a})^{(k)} - \bar{y}^{(k)}_{m,a}]$.
10: **end for**
11: **end for**
12: **if** stopping criteria are not met **then**
13: Go to step 3
14: **end if**
15: Construct a feasible solution based on the current solution
16: Call Algorithm 1 to route jump lines sequentially

In the above algorithm, $\rho$ is a pre-selected constant which is a common practice. However, a thorough observation indicates that an effective $\rho$ value should be close in magnitude to $v \times length(a)$, which is the unit cost of $y_{m,a}$. Otherwise, $w^i_{m,a}$ would yield a small fraction of (2.33) and the per-iteration change in the penalty terms $(w^i_{m,a})^{(k-1)} y_{m,a}$ would also be small.

Small changes in the penalty terms would yield little improvements in $y_{m,a}$ which in turn trigger slow PH convergence. Therefore in this chapter, we use an arc-specific value $\rho_{m,a} := \rho_0 \times (\nu \times length(a))$.

## 2.5. Computational Results

### 2.5.1. Implementation

All computational experiments in this section are performed on a server with 8GB RAM amd 2.8GHz 2220 SE Dual Core AMD Opteron Processor. The subproblems are solved by Gurobi optimization software, and the data we use to test the algorithms is derived from the New York ISO Electrical System Map, which represents the network of existing and proposed substations and transmission lines. In total, there are 692 substations and 809 transmission lines.

Through extensive computational experiments, we set $n = 180, n_1 = 35, n_2 = 10$ and $n_3 = 10$ to create a $72 \times 72$ grid network and then partition it into 180 regions. Under this setting, the major computational difficulty is caused by the large number of intersection-related variables. In Section 2.2, a binary variable $v_{k,m_1,m_2}$ is defined to capture whether two transmission lines $m_1$ and $m_2$ intersect at a particular node $k$. Due to these variables, even the much smaller size subproblems sometimes become too big to be handled in a reasonable computational time. Therefore, a hierarchical approach is employed to *first* minimize the number of turns and *then* intersections so that each subproblem can be approximately solved in seconds. To be specific, the minimization of intersections is excluded from the objective function when we solve the relaxed subproblem in each region, and is addressed later when a feasible solution is constructed.

To reduce memory needs, we use a common Lagrange multiplier for a set of similar constraints. Referring back to Section 2.4.2, $\lambda_{m,c}^{3(i)}$, $\lambda_{m,c}^{4(i)}$ and $\lambda_{m,c}^{3(j)}$, $\lambda_{m,c}^{4(j)}$ are replaced by $\eta_{m,c}^{1}$

because the corresponding constraints capture turns related to the same inter-region arc. Similarly, $\lambda_{m,c}^{5(i)}$, $\lambda_{m,c}^{6(i)}$, $\lambda_{m,c}^{5(j)}$ and $\lambda_{m,c}^{6(j)}$ are replaced by $\eta_{m,c}^{2}$. The new gradient is computed by taking the average over each component's gradient.

Since this is a large-scale problem, we focus exclusively on obtaining good quality solutions, but not on the optimality gap or other possible measures. For this reason, we stop our solution processes after 10 iterations of execution as running additional iterations does not lead to obvious further convergence. This usually results in a running time of less than 8 hours, which is considered well acceptable according to operators at NYISO control centers. Furthermore, instead of seeking an optimal solution from each subproblem, we stop solving it when the optimality gap is less than 30% or the execution time reaches 120 seconds.

### 2.5.2. Results

Given the above parameter estimates and implementation assumptions, Figure 2.4 shows an output display from the Lagrangian relxation algorithm with penalties $\alpha = \beta = \gamma = \theta = \mu = 1$ and $\nu = 1000$. It is directly drawn on a New York State map in our web-based GUI, and the geographic features on the terrain is ignored. The transmission lines are also drawn with different widths and colors to improve visibility and expedite operator's response. To validate our work, we conduct a subjective evaluation by asking 5 operators at NYISO control centers to compare our output with the current interface they are using. Three of them agreed that the reduction of clutter is apparent and the transmission lines become more traceable by human eyes with our layout. The other two voted neutral, saying that they thought the new layout is as good as the existing one. All of them liked the widely spread out substations.

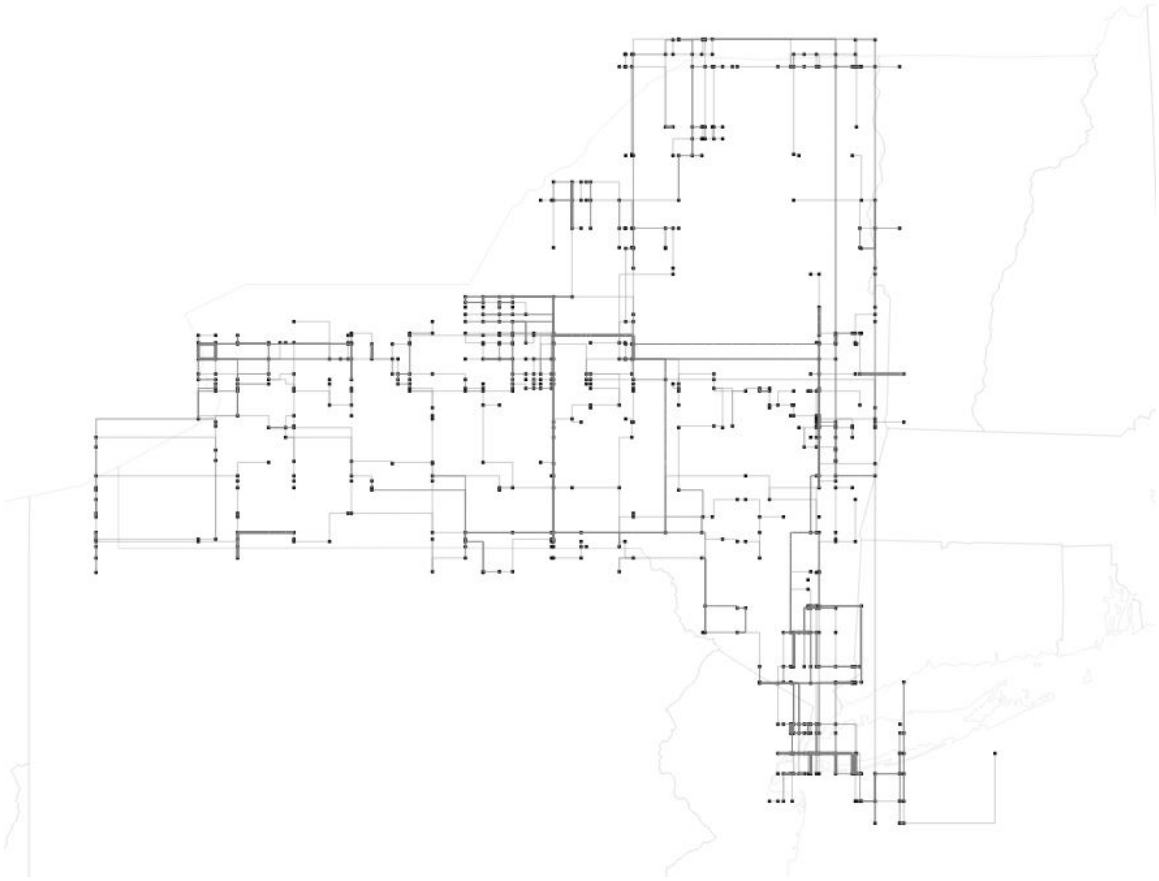

Figure 2.4. An output display from Algorithm 3

We perform computational experiments with various combinations of penalties. In the rest of this section, SR, LR and PH stand for the sequential routing, Lagrangian relaxation and progressive hedging algorithms respectively. We start by varying $\nu$ which is the penalty for loops. From the results given in Table 2.1, we can see that SR grants a significant reduction in running time compared with the iterative algorithms. It also requires less memory by avoiding a vast number of variables created in the process of solving the integer program. Therefore, SR is recommended when time and computational power are highly limited.

A major drawback of this sequential approach is the significant number of loops it generates - up to 25% of all the transmission lines. These loops would bring in unnecessary turns and congestions and thus make the transmission lines more difficult to trace on the display interface. In reality, such solutions are considered of low quality and post-processing algorithms to eliminate these loops should be developed. Also, the quality of the solution depends on the order in which the transmission lines are routed, and unfortunately, it is hard to find a good ordering. According to [2], there is no single ordering technique that consistently performs better than any other ordering method.

The numbers with asterisks in Table 2.1, which correspond to the best (smallest) objective values among the three algorithms, suggest that as the penalty for loops gradually increases, the iterative algorithms start to outperform SR.[4] When $\nu \leq 100$, SR gives the smallest objective value and is thus superior to the iterative ones. As $\nu$ keeps increasing over 200, the iterative algorithms dominate. The case when $\nu = 140$ is the trickiest since PH outperforms SR while LR does not. A thorough observation reveals that the objective values output by the three algorithms are indeed close, and thus $\nu = 140$ can be treated as a cutoff point of choosing the iterative algorithms over the sequential routing approach in our NYISO case. We expect such pattern to hold in general, but the exact values are likely to change. Therefore, if a practically implementable solution with less loops is desired, we recommend the iterative algorithms.

[4]The number of loops from the PH algorithm is not monotonically decreasing as $\nu$ increases. This is because we approximate it by the total length of transmission lines to facilitate our solution process, and certainly these two are not equivalent. Fortunately, such inequivalency does not affect our conclusion and the total length of transmission lines is indeed monotonically decreasing.

Table 2.1. Comparison of the three solution algorithms

| | **Performance Measure** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **SR** | | | **LR** | | | **PH** | | |
| $\nu$ | **time (hr)** | **loop** | **obj** | **time (hr)** | **loop** | **obj** | **time (hr)** | **loop** | **obj** |
| 1 | 0.45 | 201 | 8,650* | 5.27 | 8 | 20,319 | 4.16 | 8 | 17,410 |
| 10 | 0.44 | 200 | 10,467* | 5.24 | 3 | 23,794 | 4.20 | 6 | 21,103 |
| 100 | 0.41 | 200 | 28,527* | 6.73 | 1 | 37,820 | 4.90 | 6 | 33,177 |
| 140 | 0.41 | 200 | 36,520 | 7.15 | 0 | 37,158 | 5.21 | 10 | 35,963* |
| 200 | 0.41 | 200 | 48,508 | 7.40 | 0 | 37,436 | 5.16 | 9 | 36,381* |
| 1,000 | 0.40 | 199 | 208,518 | 8.17 | 0 | 37,609* | 5.92 | 2 | 41,522 |

Next, we compare the two iterative algorithms based on the performance measure given in Tables 2.2 and 2.3. In each row of these two tables, we assign a large penalty 1,000 to one particular objective component, and leave the other objectives less penalized with penalty 1. The columns show the results in terms of each objective. A notable remark is that the total length of all the transmission lines is not a real objective, but an approximation to the total number of loops to simplify our model.

Table 2.2. Performance analysis of LR

| | **Performance Measure** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **penalty** | **turn** | **intersec** | **overlap** | **dev (mi)** | **vio** | **len (mi)** | **loop** | **obj** | **time (hr)** |
| $\alpha = 1000$ | 244* | 132 | 764 | 4,985 | 12,394 | 14,064 | 0* | 262,275 | 6.27 |
| $\beta = 1000$ | 821 | 5* | 891 | 5,516 | 16,036 | 15,866 | 12 | 28,276 | 5.49 |
| $\gamma = 1000$ | 1,040 | 453 | 315* | 4,930 | 12,690 | 16,351 | 20 | 334,133 | 6.91 |
| $\mu = 1000$ | 774 | 130 | 794 | 5,215 | 12,212* | 15,133 | 9 | 12,218,922 | 5.16 |
| $\nu = 1000$ | 529 | 178 | 834 | 7,434 | 28,634 | 13,663* | 0* | 37,609 | 8.17 |

Table 2.3. Performance analysis of PH

| | **Performance Measure** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **penalty** | **turn** | **intersec** | **overlap** | **dev (mi)** | **vio** | **len (mi)** | **loop** | **obj** | **time (hr)** |
| $\alpha = 1000$ | 216* | 134 | 708 | 4,994 | 12,420 | 14,941 | 2* | 234,258 | 5.19 |
| $\beta = 1000$ | 759 | 8* | 806 | 4,804 | 11,110 | 14,477 | 10 | 25,489 | 4.19 |
| $\gamma = 1000$ | 874 | 419 | 330* | 4,780 | 11,566 | 15,470 | 18 | 347,657 | 7.32 |
| $\mu = 1000$ | 683 | 151 | 780 | 5,214 | 10,220* | 15,006 | 10 | 10,226,838 | 4.09 |
| $\nu = 1000$ | 471 | 167 | 854 | 7,804 | 30,226 | 13,653* | 2* | 41,522 | 5.92 |

As indicated by the numbers with asterisks in Tables 2.2 and 2.3, assigning a larger penalty to a particular objective component gives a solution with smaller value in that regard. The only exception happens for the number of loops because clearly there is a tie between $\alpha = 1000$ and $\nu = 1000$. This implies that the approximation we used is not equivalent to the original objective. Fortunately, despite the tie, $\nu = 1000$ still gives a solution with the least number of loops, and hence we can say that such inequivalency does not harm the applicability of our approximation.

In Tables 2.2 and 2.3, for the same combination of penalties, LR and PH output very similar results. It is not easy to distinguish them by a clear cutoff value as what we did between SR and the iterative algorithms. However, a thorough observation still reveals a few guidelines regarding how to select an appropriate algorithm based on the computational results at hand. As shown in the first, second and fourth rows of each table, when a large penalty is assigned to *turns*, *intersections* or *violations*, PH delivers a better solution within a shorter time, and thus outperforms LR. Similarly, when a large penalty is assigned to *overlaps*, LR performs better. When loops are penalized with $\nu = 1000$, a trade-off between the quality of the solution and

running time is involved as LR reaches a better solution while PH runs faster. We do not consider the case when $\theta = 1000$, because the total deviations had already been minimized when we assigned each substation into a region by solving (2.12) - (2.16) no matter what combination of penalties is selected to solve the subproblems. Thus, the total deviations in the final solution is not completely determined by the ranking of the penalties, and should not be compared in the same way as the other objective components.

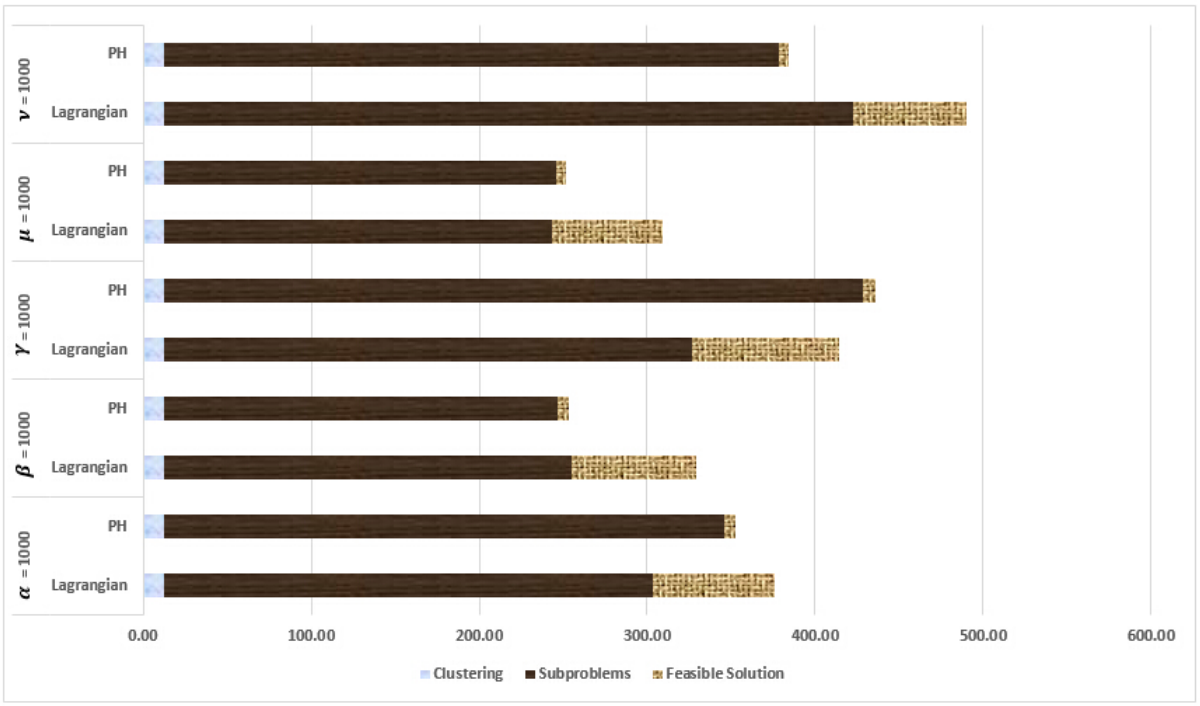


Figure 2.5. Running time of major components in the solution process

As we can see from Figure 2.5, the majority of the running time is spent on solving the independent subproblems. Therefore, parallel computing has a great potential to reduce the running time by solving the subproblems simultaneously. Moreover, as PH spends a larger proportion of time on subproblems, it would benefit more from parallel computing.

## 2.6. Summary and Future Research

In conclusion, we have proposed a mixed-integer model to optimize the layout of visual elements on a power transmission display map and developed three algorithms to effectively solve this NP-hard problem. Besides direct applications in power transmission control centers or more generally in display technology, our modeling and solution approaches can be extended to other complex networks such as transportation and supply chain systems. An interesting example is drone delivery. To make it possible in the future, logistics companies have to design routes to avoid intersections and overlaps to mitigate risk of collision and avoid loops to save time and energy costs. They also need to decide the candidate locations to build control centers and parking facilities.

There are many related topics worth of further effort. First, our iterative algorithms cannot explicitly handle transmission lines which are routed across more than two regions. Thus, a more general approach that is able to resolve this difficulty should be potentially considered. Second, we propose a heuristic shortest path algorithm to take the minimization of line turns and intersections into consideration. Although a virtual layer method like [23] and [68] could be applied to handle the minimization of turns in lattice graphs, to the best of authors' knowledge, an optimal approach to take care of both does not exist and is thus left for future research.

CHAPTER 3

# Solving Attractiveness-based Stochastic Fleeting by MapReduce

## 3.1. Motivation and Literature Review

Airlines around the world continue to face increasing capital and operational costs. As competition intensifies, they have been forced to cut costs and uphold revenue by utilizing their equipment capacity more efficiently to accommodate passenger demand. This is generally known as the *fleet assignment problem*, which deals with assigning aircraft of different capacities to the scheduled flight legs based on availabilities, costs, potential revenue and itinerary-based passenger demand. Due to the large number of scheduled flights and the dependency of other airline operations (e.g., crew scheduling) on fleet assignment, solving the underlying *fleet assignment model* (FAM) is never an easy task. What further complicates the problem are:

- **Stochastic demand**: the fleet assignment decision is made 10 - 12 weeks in advance of departure, and at such an early stage, the level of market demand uncertainty is usually high. Deterministic models using average demand are thus inadequate to reflect the final demand realization.
- **Network effects**: to have a more accurate estimate of passenger flow and revenue, spill and recapture effects need to be properly addressed in an itinerary-based FAM. In reality, passengers spilled from an itinerary usually compare all the available options in the market and choose the most attractive one in terms of price, departure and arrival time, number of stops, total duration, and other factors that affect their preferences.

As a result, airlines usually solve an initial FAM based on early demand forecasts, and later swap aircraft assignments in response to demand variations (demand driven dispatch). In terms of spill and recapture, they either assume a constant recapture rate or penalize any spill or unmet capacity in the objective function.

In this chapter, we present a two-stage stochastic model to explicitly consider potential market scenarios. The first stage is a basic fleet assignment model which decides the assignment of fleet types to flight legs. The second stage then finds the optimal passenger flow on available itineraries for each scenario based on the assigned capacities and the demand estimated for each itinerary. Our model inherits the attractiveness-based spill and recapture framework from [71] which does not consider demand stochasticity, and by doing so, any spilled passenger is recaptured with a probability proportional to the attractiveness of an alternative itinerary. Another novelty of the proposed model is the integration with a production simulator which generates market scenarios and solves the corresponding second stage problems by considering bookings of multiple periods over the past year and advanced options such as up- and down-sells. This brings our model closer to a point of potential integration with airlines' decision support systems and making immediate impact. Unfortunately, due to the size of the model and the number of scenarios under consideration, solving it directly is computationally challenging. Hence, developing effective solution methodologies is also a focus of this research.

Studies on fleet assignment can be traced back 30 years. Abara [1] was one of the first researchers to examine realistically-sized FAM using a connection network. In contrast, Berge and Hopperstad [9] and Hane et al. [30] were among the first researchers to use the time-space network, which has quickly become the dominant method to formulate FAM. A notable effort in modeling fleet assignment is the itinerary-based FAM proposed in [7], which combines the basic

fleet assignment model with a passenger mix model to explicitly capture network effects. Unlike our attractiveness-based approach, they assume that the rate of recapture is calculated through a Quantitative Share Index (QSI) and remains constant regardless of what options/itineraries are available in the market.

The works that are most closely related to ours are [63] and [45]. Sherali and Zhu [63] propose a two-stage stochastic model in which the first stage conducts an initial fleet assignment by making only family-level assignments to flight legs, while the second stage performs subsequent family-based type-level assignments to accommodate passenger demand for each scenario. They conduct polyhedral analysis and develop a Benders decomposition-based solution approach. In our work, we do not consider the family-level assignment separately but complete all the assignment tasks in the first stage. The second stage of our model focuses on attractiveness-based network effects and is a more complicated production version that considers additional features such as up- and down-sells and fare estimation. In terms of the solution approach, we develop Benders decomposition-based algorithms tailored for a distributed computing environment to further reduce the computational time. Listes and Dekker [45] also take the stochasticity of demand into consideration, but they focus on fleet composition[1] rather than fleet assignment and they do not consider spill and recapture. Their solution approach is based on progressive hedging (PH). Due to the size and complexity of individual scenarios, they pursue the strategy of first finding a solution to the linear relaxation of the model, and then using a simple rounding procedure to obtain integer results. In this chapter, we also propose a heuristic algorithm based on PH, but what makes our work significantly different from theirs are: 1) we incorporate a Benders decomposition framework into the traditional PH; 2) we do not solve a

[1]Fleet composition: given a set of aircraft types, the fleet composition problem is to determine the optimal number of aircraft of each type to be used to maximize the profit.

linear relaxation of the model. To address the computational challenge, we employ a distributed framework named MapReduce. We decide to use MapReduce instead of other more traditional and in a certain sense more powerful architectures such as MPI because Hadoop and MapReduce are now ubiquitous in practice. The majority of companies have Hadoop installations and data scientists using them. This is definitely not the case for MPI.

The three algorithms developed in this chapter are distributed versions of the classic Benders decomposition and PH algorithms. In our Benders decomposition-based approaches, all the scenario-dependent subproblems (i.e., the second stage of our model) are solved and the corresponding Benders optimality cuts are generated simultaneously on cluster computers to reduce the computational difficulty of having multiple scenarios. Our PH algorithm works similarly by solving the scenario problems in parallel. However, unlike the traditional PH approach, we solve each scenario problem by a Benders decomposition algorithm and add the Benders cuts generated in previous iterations as extra constraints to the next iteration. With such preservation of Benders cuts and the presence of binary variables in our model, this approach is only a heuristic. We evaluate and compare these algorithms using data from a medium-size airline, and our computational results establish feasibility in using the two-stage stochastic model and the aforementioned distributed methodologies to solve FAM. A 10 - 15% increase in profitability is observed, and the optimality gaps from all the algorithms are less than 12%.

The contributions of this research are as follows.

(1) According to authors' best knowledge, this is the first effort to apply MapReduce to address the computational challenge of a large-scale two-stage stochastic program. An extension to multi-stage is straightforward.

(2) This is the first time that the stochastic nature of passenger demand is explicitly addressed under an attractiveness-based spill and recapture framework. The two-stage stochastic model we formulate is an improvement over the deterministic version proposed in [71].

(3) The incorporation of a Benders decomposition algorithm into PH, and preserving Benders cuts from previous iterations as additional scenario problem constraints is a novel modification of the conventional progressive hedging approach.

(4) We integrate the proposed algorithms with a production simulator developed by Sabre Airline Solutions to solve the second stage, and thus make them immediately applicable to industrial practitioners.

The rest of this chapter is organized as follows. In Section 3.2, we formulate the two-stage stochastic model with attractiveness-based spill and recapture effects. Section 3.3 presents the three distributed algorithms based on MapReduce. Section 3.4 gives the findings from our computational experiments and presents sensitivity analyses. Finally, we conclude in Section 3.5.

## 3.2. A Stochastic Formulation of the Problem

In order to take the stochastic nature of passenger demand into explicit consideration, we formulate a two-stage stochastic model that considers several potential market scenarios. The entire formulation relies on the notion of an embedded time-space network, which is typically used in formulating FAM and representing flight schedule. To learn more about this network and its applications, we refer readers to, for example [61].

The first stage of our model assigns aircraft to the scheduled flight legs on a daily basis so that the following criteria are met: 1) Each flight leg is covered by exactly one fleet type; 2) Aircraft arriving at an airport at a particular time must leave that airport at some time later to ensure the same daily schedule and avoid deadheading; 3) Only available aircraft of each fleet type are used in the assignment. These criteria are usually referred to as *cover constraints*, *conservation of flow constraints* and *aircraft count constraints*. Since this part has been repeatedly formulated in the classic fleet assignment literature (e.g., [7]), it is hence omitted in this chapter. The second stage then utilizes the assigned capacity to accommodate the itinerary-based demand for each scenario, and adopts the attractiveness-based network model of Wang et al. [71] to handle the spilled passengers. We incorporate this framework because:

(1) Modeling spill and recapture is essential for passenger flow estimation and capacity planning, and hence relates to the fleet assignment problem when matching fleet types with different capacities to flight legs.

(2) In reality, (spilled) passengers constantly change their preferences based on all the available itineraries in the market. This should be reflected in modeling.

The following notation is used. An itinerary always has the underlying market. Sometimes we show this relationship explicitly in order to stress it.

**Sets**:

- $L$: set of scheduled flight legs, indexed by $l$.
- $K$: set of fleet types, indexed by $k$.
- $M$: set of all markets, indexed by $m$.
- $\Pi_m^{HA}(l)$: set of all itineraries of the host airline ($HA$) that include flight leg $l$ in market $m$, $m \in M$, $l \in L$.

· $S$: set of scenarios in terms of demand realization, indexed by $s$.

**Decision Variables**:

· $x_{lk} = 1$ if flight leg $l$ is flown by fleet type $k$; 0 otherwise.

· $q_i^s$: market share of itinerary $i$ in market $m$ under scenario $s$, $i \in \Pi_m^{HA}$, $m \in M$, $s \in S$.

· $q_m^s$: market share of all the itineraries offered by other airlines in market $m$ under scenario $s$, $m \in M$, $s \in S$.

· $q^s$: vector of market share over all markets under scenario $s$, $s \in S$. That is, $((q_i^s)_i, q_m^s)_m$.

**Parameters**:

· $Cap_k$: seat capacity of fleet type $k$, $k \in K$.

· $c_{lk}$: cost of assigning fleet type $k$ to flight leg $l$, $k \in K$, $l \in L$.

· $p^s$: probability for realization of scenario $s$, $s \in S$.

· $D_m^s$: total demand realization for market $m$ under scenario $s$, $m \in M$, $s \in S$.

· $D^s$: vector of demand realizations over all markets under scenario $s$, $s \in S$. That is, $D^s = (D_m^s)_m$.

The general form two-stage stochastic model reads as follows, where $x = (x_{lk})_{l,k}$.

$$\min \sum_{l \in L} \sum_{k \in K} c_{lk} x_{lk} - \sum_{s \in S} p^s Q(x, D^s) \tag{3.1}$$

s.t. *Cover Constraints*

*Conservation of Flow Constraints*

*Aircraft Count Constraints*

where, for each $s \in S$, we have

$$Q(x, D^s) = \max_{q^s} \; r(D^s, q^s) \tag{3.2}$$

$$\text{s.t.} \quad e(D^s, q^s) \leq \sum_{k \in K} x_{lk} Cap_k \qquad l \in L \tag{3.3}$$

$$q^s \in V \tag{3.4}$$

The objective function (3.1), together with (3.2), minimizes the total expected cost, or equivalently, maximizes the total expected profit. The first stage includes the typical cover constraints, flow balance constrains and aircraft count constraints to make the assignment of fleet types to flight legs. The second stage is a passenger mix model with attractiveness-based spill and recapture. The objective is to find the optimal passenger flow on all the available itineraries, subject to the limited capacity assigned to the flights and the demand estimated for each itinerary. Unlike [71] or any traditional approaches, in this chapter, we use a sophisticated production version of $Q(x, D^s)$ where bookings over a year are accumulated and advanced options such as up- and down-sells are captured. A simplistic model is given next.

### 3.2.1. A Simplistic Second Stage Model

The following additional notation is used in the simplistic model.

- $f_i^s$: average fare estimate for itinerary $i$ under scenario $s$, $i \in \Pi_m^{HA}$, $s \in S$.
- $A_i$: attractiveness of itinerary $i$ in market $m$, $i \in \Pi_m^{HA}$, $m \in M$. According to [71], the attractiveness can be quantified by $e^U$, where utility $U$ is a linear combination of a

series of attributes such as departure time, arrival time, number of stops, total duration and so on.

- $A_m$: attractiveness of all the itineraries from other airlines in market $m$, $m \in M$. This represents passenger utility of flying with a different airline.

The model hence reads as:

$$Q(x, D^s) = \max_{q^s} \sum_{m \in M} D_m^s \sum_{i \in \Pi_m^{HA}} f_i^s q_i^s \tag{3.5}$$

$$\text{s.t.} \quad \sum_{m \in M} D_m^s \sum_{i \in \Pi_m^{HA}(l)} q_i^s \leq \sum_{k \in K} x_{lk} Cap_k \qquad l \in L \tag{3.6}$$

$$\sum_{i \in \Pi_m^{HA}} q_i^s + q_m^s = 1, \qquad m \in M \tag{3.7}$$

$$A_m q_i^s \leq A_i q_m^s, \qquad i \in \Pi_m^{HA},\ m \in M \tag{3.8}$$

$$q^s \geq 0 \tag{3.9}$$

Objective function (3.5) maximizes the revenue captured from all the itineraries of the host airline. Constraints (3.6) impose that the total demand captured on all itineraries that include leg $l$ cannot exceed the capacity assigned to leg $l$. Constraints (3.7) and (3.8) ensure that the probability of recapturing the spilled demand is proportional to the attractiveness of the alternative itinerary. This model modifies the work in [71], and is only used to explain the concept of attractiveness-based network effect.

## 3.3. Algorithms

The two-stage stochastic model developed in Section 3.2 has mixed binary variables in the first stage and continuous variables in the second stage. Since the number of daily flights/itineraries for a major airline can easily reach several thousands, and the proposed model also explicitly considers multiple scenarios, it turns out to be a very hard problem. Efficient solution methodologies hence need to be developed to mitigate the inherent combinatorial burden and to supply a practically good solution within a reasonable computational time.

In this section, we first introduce a decision support simulator named Airline Planning and Operations Simulator (APOS), and explain how to use it to generate random demand realizations and get $Q(x, D^s)$. We then present three distributed algorithms - two Benders decomposition-based algorithms and one progressive hedging-based algorithm - to solve multiple scenarios concurrently to reduce the overall computational time. All the distributed algorithms developed in this chapter are based on the MapReduce framework.

### 3.3.1. Integration of APOS

We start by formally introducing the Airline Planning and Operations Simulator (APOS). Airlines use various methodologies in their planning processes. The most reliable and widely used method to assist decision making is simulation that allows to replicate actual customer behavior in the controlled environment and to carry out multiple experiments on the same set of inputs. APOS, provided by Sabre Airline Solutions, was originally designed to fulfill these purposes for the sake of revenue management. In particular, it reads historical customer booking information to generate forecast streams, which we call potential demand realizations in the context of this research, and then performs optimization algorithms on the generated streams to achieve

the highest expected revenue. With slight modification, the simulator is able to provide the following two crucial functions which will be repeatedly used in designing our overall algorithms.

(1) Given a random seed, APOS generates a particular market-level demand realization or scenario.

(2) Given a feasible fleet assignment solution (i.e., a feasible first stage solution), APOS solves the attractiveness-based passenger mix model in the second stage and returns the highest revenue, $Q(x, D^s)$.[2] Meanwhile, it also generates other outputs, such as the dual solution, to reveal more information about the second stage model it solves.

Therefore, as shown in Figure 3.1, we incorporate APOS into the two-stage stochastic model as a blackbox function to generate multiple scenarios and solve the corresponding second stage problem (3.2) - (3.4) for any given scenario. Being a blackbox, how the simulator actually operates and what algorithms it employs are out of the scope of this research, and thus omitted.

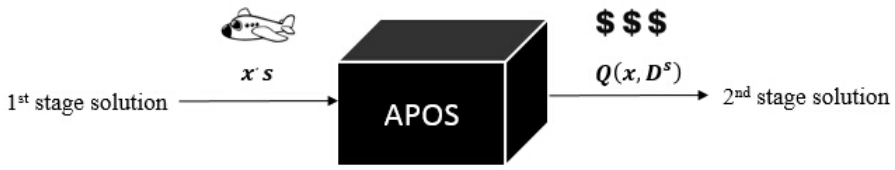


Figure 3.1. Integration of APOS as a blackbox

### 3.3.2. Three MapReduce-based Decomposition Algorithms

In this section, we propose three decomposition algorithms to partition the problem into multiple smaller subproblems - two primal decomposition methods based on Benders decomposition and one dual decomposition method based on progressive hedging. Since all these algorithms

[2]APOS offers a function mode named *SBLP*. It is developed based on the passenger mix model in [71], but also incorporates many practical considerations (up- and down-sells, cargo capacity, etc).

involve solving multiple independent subproblems with similar structures, parallel computing has great potential to reduce the computational time by solving the subproblems simultaneously. Thus, all the algorithms to be exhibited in this section are developed based on the distributed framework named MapReduce.

MapReduce is a software framework to submit and process parallelizable applications which deal with vast amount of data on clusters of computers. In this chapter however, we leverage this popular "big data" technology to solve a large-scale stochastic program. A MapReduce program is typically composed of **mappers** which turn each data record into a key-value pair and **reducers** which perform a summary operation. In our case, a mapper instructs a number of reducers which scenario to solve, and then each reducer calls APOS to solve the second stage model of the assigned scenario. Further details about the implementation are listed in Section 3.4.1. The input to the algorithm includes a list of scenario ID's (e.g., $1, ..., |S|$) and other standard input to FAM.

**Two MapReduce-based Benders Decomposition Algorithms.**

Benders decomposition is a popular method for solving certain large-scale optimization problems, such as the two-stage stochastic programming model formulated in this chapter. The solution process of a typical Benders algorithm alternates between a master problem and subproblems corresponding to different scenarios. In our case, the master problem is solved by branch-and-bound in the space of the first stage variables, and each subproblem is solved by executing APOS with a constant seed specifying the scenario. The corresponding Benders

optimality cut is then constructed by utilizing the dual information as in the classic Benders algorithm [8]. We omit the details of the classic Benders algorithm since it has been well documented in a variety of literature.

In our MapReduce-based algorithms, all the scenario-dependent subproblems are solved in parallel and the associated Benders cuts are generated separately. As we can observe from Figure 3.2, with all these cuts available, the Benders algorithm can proceed with either of the following two directions: 1) adding all the cuts to the master problem and repeat the solution process until some stopping criterion is met; 2) aggregating all these cuts into a single cut by using the expected value of coefficients and then adding it to the master problem to continue the solution process. The following Algorithms 1 and 2 are hence developed in line with these two directions.

**Algorithm 5** The MapReduce-based Multiple-cuts Benders Algorithm

1: Initialization:
   $LB := -\infty$, $UB := \infty$
2: Solve the first stage master problem
3: Update $LB$
4: **while** stopping criteria not met (e.g., $UB - LB > \varepsilon$) **do**
5: Call MapReduce to solve each second stage subproblem and generate Benders cuts $C_1, ..., C_{|S|}$
   **Mapper:** Assign one scenario ID to each reducer
   **Reducer:** Run APOS to solve the second stage of the assigned scenario $s \in S$, and generate $C_s$
6: Retrieve $C_1, ..., C_{|S|}$
7: Update $UB$
8: Add $C_1, ..., C_{|S|}$ to the master problem
9: Solve the first stage master problem
10: Update $LB$
11: **end while**

In the above algorithms, the parallelization is done through the MapReduce framework and APOS is treated as an integrated function, which can be repeatedly called in the reducing phase.

**Algorithm 6** The MapReduce-based Aggregate-cut Benders Algorithm

1: Algorithm 5 steps 1 - 7
2: Aggregate $C_1, ..., C_{|S|}$ into a single cut $\bar{C}$ using the expected coefficients
3: Add $\bar{C}$ to the master problem
4: Algorithm 5 steps 9 - 11

The MapReduce framework consists of mappers and reducers, in which mappers take the scenario ID's (i.e., $1, ..., |S|$) as input and emit a unique ID to each reducer, and each reducer takes the solution of the master problem as input via distributed cache (see Section 3.4.1 for details) and executes APOS to solve the second stage subproblem based on the ID assigned to it. The two algorithms only differ in the way of handling Benders cuts and thus have the same complexity.

**A Progressive Hedging Algorithm based on MapReduce.**

When confronted with a two-stage stochastic program for which there exist effective algorithms to solve individual scenarios, the progressive hedging (PH) method is another popular approach. However, in the context of our model, it is not possible to explicitly solve the (deterministic) problem corresponding to each scenario because APOS is considered a blackbox and thus we do not know how it generates the scenario and what model and algorithms it employs. To get around this difficulty, a natural starting point is to construct a Benders decomposition framework similar to what we developed previously so that the second stage of each given scenario can be solved by APOS without knowing the model or the solution process. That being said, in each iteration of the to-be-proposed PH algorithm, Benders decomposition is employed for solving scenario problems.

A MapReduce-based approach is again proposed to solve the independent scenario problems in parallel. Mappers still read the scenario ID's as input and then feed each reducer a unique ID. However, for any given scenario $s \in S$, the reducer no longer only solves the second stage problem and generates a single cut, but uses Benders decomposition to iteratively solve the complete model specified by scenario $s$. That being said, if we let $P_1^s$ and $P_2^s$ be the first and second stages of the scenario problem $P^s$, where $s \in S$, the output from step 5 of Algorithms 5 and 6 is a Benders optimality cut corresponding to $P_2^s$ while the output from the same step of the following PH algorithm is a solution to $P^s$. The PH algorithm is essentially a sequential version of Algorithms 5 and 6 performed on individual scenarios.

A potential issue with the above solution approach is that each scenario problem, although much smaller than the overall problem, is still not easy to solve to optimality in a relatively short time. To address this challenge, we modify the PH algorithm in such a way that the integrated Benders solution process in every PH iteration is terminated earlier before an optimal solution is found and all the Benders cuts generated from previous iterations are retained and added to the next iteration. It is easy to see that these cuts are valid since they are based on the same scenarios with only objective function being varied. This leads to the following Algorithm 7, taking $\rho$ as a penalty factor.

**Algorithm 7** The MapReduce-based Progressive Hedging Algorithm

1: Set $v := 0$
2: Initialize $\bar{x}^{(0)}$ and $(w^s)^{(0)}$
3: **while** stopping criteria not met **do**
4: Set $v := v + 1$
5: Call MapReduce to solve each scenario problem in parallel and obtain $(x^s)^{(v)}$
**Mapper:** Assign one scenario ID to each reducer
**Reducer:** Use the Benders decomposition algorithm to solve the assigned scenario $s \in S$:

$$(x^s)^{(v)} := \arg\min_x \left( \sum_{l\in L}\sum_{k\in K} c_{lk}x_{lk} - Q(x, D^s) + \sum_{l\in L}\sum_{k\in K} (w^s_{lk})^{(v-1)}x_{lk} + \frac{\rho}{2}\sum_{l\in L}\sum_{k\in K} \|x_{lk} - \bar{x}^{(v-1)}_{lk}\|^2 \right)$$

**s.t.** $1^{st}$- and $2^{nd}$-stage constraints and Benders cut constraints from iterations $v-1, v-2, ..., 0$
6: Retrieve $(x^s)^{(v)}$ and Benders cuts from each scenario
7: Set

$$\bar{x}^{(v)}_{lk} := \sum_{s\in S} p^s (x^s_{lk})^{(v)},\ (w^s_{lk})^{(v)} := (w^s_{lk})^{(v-1)} + \rho\left((x^s_{lk})^{(v)} - \bar{x}^{(v)}_{lk}\right)$$

8: Add retrieved Benders cuts to corresponding scenario problems
9: Substitute each $((x^s)^{(v)})$ into (3.1), and set

$$x^{(v)} := \arg\min_s \sum_{l\in L}\sum_{k\in K} c_{lk}(x^s_{lk})^{(v)} - \sum_{s\in S} p^s Q\left((x^s)^{(v)}, D^s\right)$$

10: **end while**

In the above algorithm, since all the scenario-dependent solutions from Step 6 are feasible, we substitute each of them into (3.1) for cost evaluation and pick the one with the minimal cost as an overall primal solution. We also tried other more sophisticated ways of getting an overall solution, for example, solving $x^{(v)} := \arg\min_s \sum_{l\in L}\sum_{k\in K} -\omega_{lk}x_{lk}$ subject to the cover constraints, conservation of flow constraints and aircraft count constraints. The weight $\omega_{lk}$, $l \in L$, $k \in K$ is determined by how many times fleet type $k$ is assigned to flight leg $l$ across all scenario solutions. The more often $k$ is assigned to $l$, the larger the weight would be, thus leading to a smaller coefficient in the above expression and a higher chance of $x_{lk} = 1$ in the overall

solution. In this chapter, we go with the simple approach in Algorithm 7 as it gives the best solution. Graphical illustrations of the three algorithms are given in Figures 3.2 and 3.3.

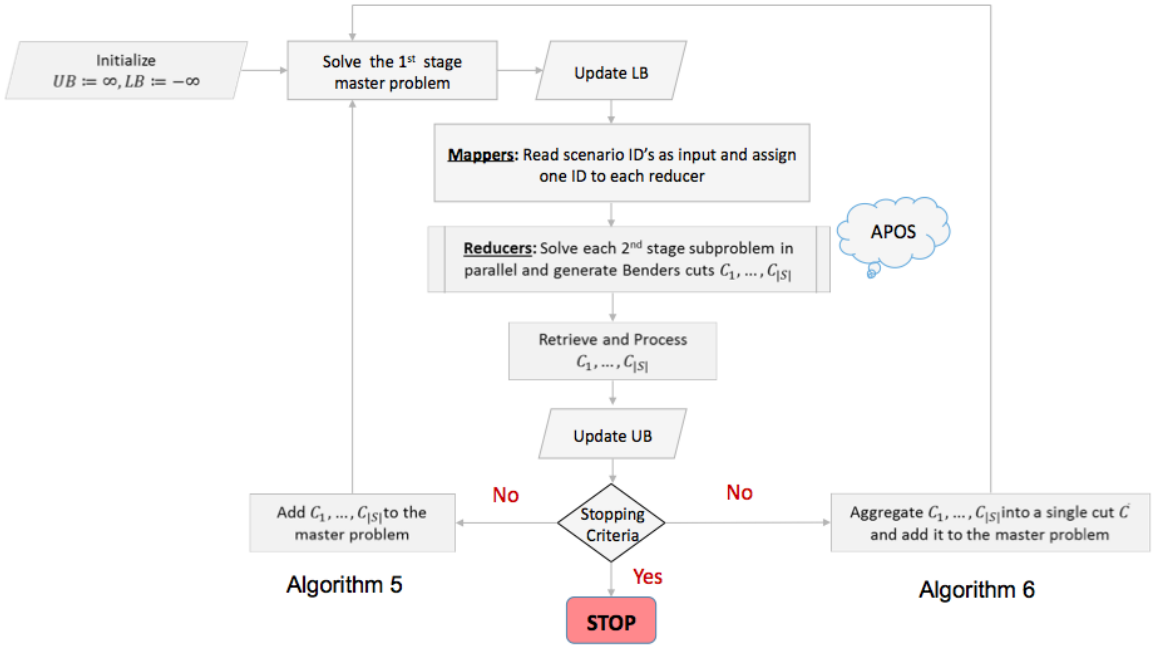


Figure 3.2. Solution process flow of the two MapReduce-based Benders decomposition algorithms

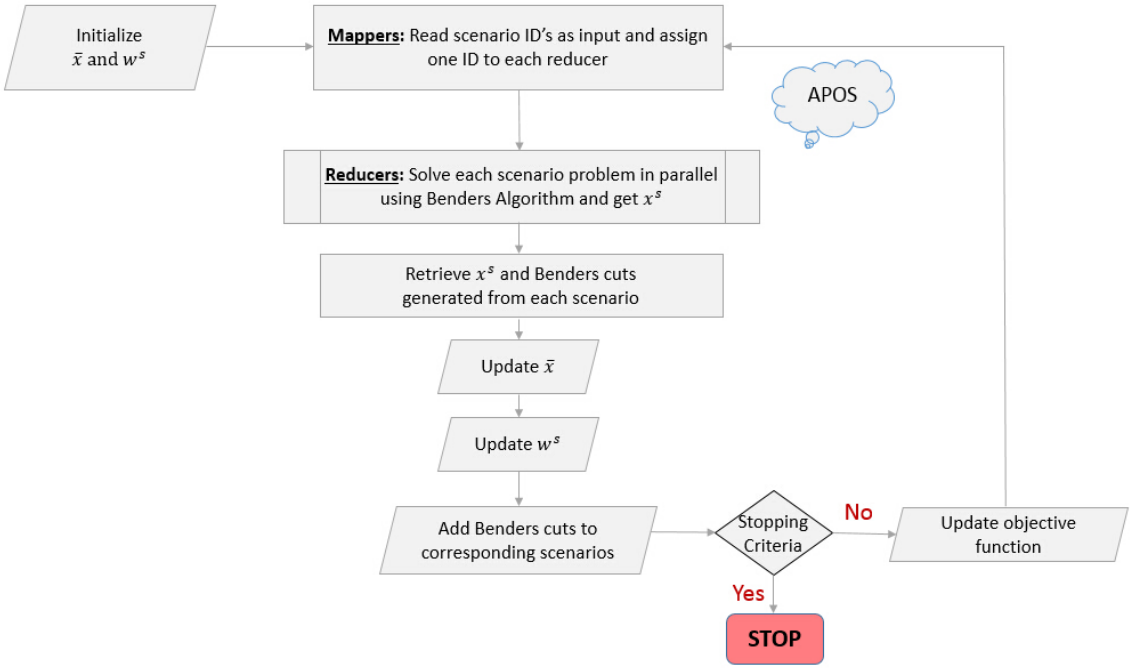


Figure 3.3. Solution process flow of the MapReduce-based PH algorithm

## 3.4. Computational Experiments

### 3.4.1. Implementation

All computational experiments are conducted on a Hadoop cluster. The cluster consists of four 2.2GHz Xeon CPU's, each having 8 cores and hence resulting in 32 cores for parallel program execution. Each of the four servers has 32 GBytes of main memory. They serve as data nodes and there is a separate named node. Cloudera version 5 is the backbone Hadoop installation. All the optimization problems are solved by IBM ILOG CPLEX Optimization Studio.

To test the performance of the proposed algorithms, we use data from a medium-size airline which involves 132 international flights and 20 fleet types. Since we have 32 cores, 32 scenarios are generated by APOS to reflect market demand fluctuations. The input to the simulator includes information of equipments (fleet types), flights, markets and days when booking information is updated and the forecast is reoptimized. In our experiments, these days are specified as 366, 226, 107, 57, 35, 23, 17, 13, 9, 5, 3 and 1, so that bookings of various time periods over the past year are accumulated to capture the cyclic nature of passenger demand. All the data is provided by Sabre Airline Solutions, and the utility coefficients in attractiveness are calibrated by APOS.

By means of extensive numerical experiments, we notice that 5 hours of running time is enough to get a good quality solution from the proposed algorithms, and running additional hours does not lead to obvious further convergence. Therefore, for the rest of this section, running time not exceeding 5 hours is imposed as a stopping criterion for Algorithms 5 - 7. The resulting optimality gaps are all around 10%. We also set the relative MIP gap tolerance (a Cplex parameter) to $10^{-2}$ to solve the first stage master problem faster[3]. Since all three algorithms involve solving a Benders master problem somehow, this setup is crucial in terms of expediting the overall solution approaches. Finally, instead of seeking an optimal solution in every iteration of Algorithm 7, we stop the solution process of each scenario problem after 15 minutes of execution or after 20 cuts are generated. The motivation of doing this is because: 1) with the presence of binary variables, our PH approach is only a heuristic, and thus an optimal solution in each iteration does not guarantee any overall convergence; 2) 15 minutes

[3]The default setting of the MIP tolerance gap in Cplex is $10^{-4}$.

of execution or 20 Benders iterations usually leads to a reasonably good solution for a single scenario problem, with the corresponding optimality gap around 30%.

In Algorithms 5 - 7, both mappers and reducers are implemented in Python using Hadoop Python streaming. Since Hadoop Python API does not support user-defined partitioning, to ensure each reducer only processes one scenario, we mimic the partitioning function by: 1) randomly generating 32 java string hashcodes with equal length; 2) using them as scenario IDs. As a result, the default partitioner, which partitions the data using Java string hashcode as hash key, splits the output from the mappers into 32 segments (scenarios) and gives them to 32 reducers.

From the implementation standpoint, passing information to or retrieving information from MapReduce is not straightforward and hence needs further discussion. There is an overarching external scripts that executes the highest level loop. In each outer iteration of Algorithms 5 and 6, we pass the first stage master problem solution to each reducer using Hadoop Distributed Cache. Specifically, in each iteration we write the solution to a text file and then cache it so that it is accessible to all the data nodes (reducers). Once reducers finish solving all the subproblems, information to construct Benders optimality cuts (i.e., dual information, coefficient matrices and right-hand sides) is written to the Hadoop Distributed File System (HDFS). To retrieve these cuts and add them to the master problem for next iteration, the external script copies the output of reducers from HDFS to the local filesystem for further processing. In Algorithm 7, Benders cuts and $(x^s)^{(v)}$ corresponding to each scenario are retrieved in a similar way through HDFS. However, the information passed to the reducers is significantly different since each reducer no longer only solves the second stage subproblem and generates a single cut, but employs Benders decomposition to solve a complete model (with both first and second stages). For this reason, in

every iteration of Algorithm 7, each reducer requires a scenario-dependent master problem and the additional Benders cut constraints. Information to construct these models is written to a text file and sent as distributed cache to make it accessible to all the reducers. No LP or MPS files are transferred in the above implementation: only customized text files are used. Finally, APOS is a C++ library which is called from a reducer through the standard Python-to-C++ interfacing.

### 3.4.2. Computational Results

In order to examine the benefits of the proposed algorithms, we take the airline's current fleeting decision as a benchmark solution. The most important computational results are presented in Table 3.1, in which a positive sign indicates an increase in the objective value or a decrease in profit, and a negative sign indicates the opposite. These principles are followed by all the tables and figures presented in this section. All solutions including the benchmark solution are compared with respect to the expected cost based on the 32 generated scenarios. As we can see, the solutions from the proposed algorithms all yield cost reduction against the benchmark, and a 10 - 15% improvement is often considered significant for airlines with tight margins. The two Benders decomposition-based algorithms output similar objective values and optimality gaps. The solution approach with 32 Benders cuts added to the master problem in each iteration (Algorithm 5) gives a smaller objective value, while the approach adding a single aggregate cut (Algorithm 6) renders a narrower optimality gap. Between these two, we also see more Benders iterations in Algorithm 6, which is intuitive since we add fewer constraints to the master problem in every iteration and that usually leads to a simpler problem and a faster solution process. Algorithm 7 yields the best solution in Table 3.1. As shown in Figure 3.4, a solution superior to those given by the Benders algorithms can be achieved within 3 hours. We also

observe a large discrepancy in the objective values favoring Algorithm 7 in the first two hours of execution. Thus, the superiority of this Benders-incorporated PH approach is demonstrated and it is recommended when a quick solution is desired.

Table 3.1. Comparison of the three MapReduce-based algorithms

| | Obj. vs. Benchmark$_{32}$ | Optimality Gap | # of Benders Iterations |
|---|---|---|---|
| Algorithm 5 | -11.5% | 11.9% | 86 |
| Algorithm 6 | -10.2% | 10.2% | 189 |
| Algorithm 7 | -12.4% | — | 5,465 |

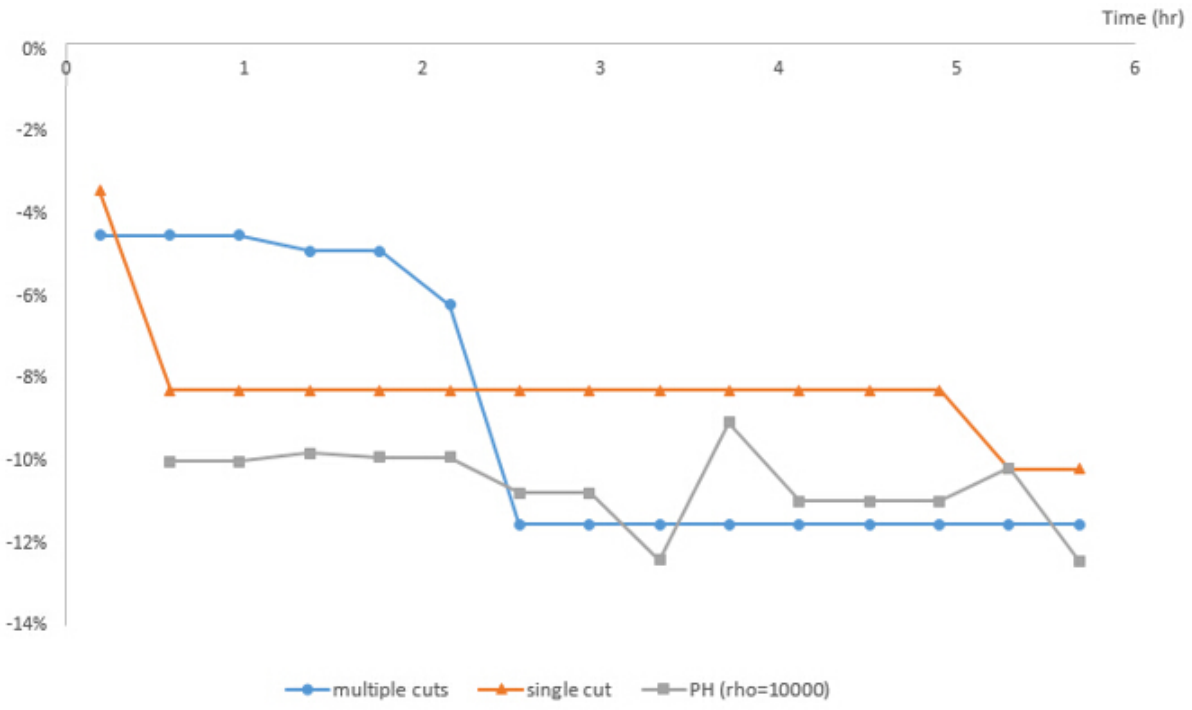


Figure 3.4. Obj. vs. Benchmark$_{32}$ in 5 hours

Due to the the presence of binary variables in our model, Algorithm 7 is simply a heuristic and the optimality gap is theoretically intractable. Furthermore, since we conduct 32 independent Benders decomposition processes in each iteration of Algorithm 7, a large number of Benders iterations (i.e., over 5,000) is expected.

Next, we evaluate both the benchmark solution and the solutions from the three algorithms on 1,000 scenarios for a more accurate cost estimate. The evaluation process is simple: given a solution $x_{lk}$, $l \in L$, $k \in K$, we are able to calculate $\sum_{l \in L} \sum_{k \in K} c_{lk} x_{lk}$ in (3.1); then we pass $x_{lk}$ and 1,000 distinct scenario IDs sequentially to APOS to let it solve for each $Q_s(x, D^s)$, $s \in S$ with $|S| = 1,000$. With $p^s$ being a known parameter, we can easily get the objective value corresponding to each solution. The results are given in Table 3.2, where *Obj.* is obtained based on 32 scenarios and *Eval.* is evaluated based on 1,000 scenarios. As we can see, the solutions from the proposed algorithms still yield improvement over the benchmark, and Algorithm 7 still outputs the best result, followed by Algorithm 5 and then Algorithm 6. We also notice that all three objective values increase compared to those evaluated based on 32 scenarios. This is intuitive since our solutions are derived by solving the two-stage stochastic program with 32 scenarios, they are unlikely to be optimal under the additional scenarios. Therefore, the percentage increase shown in the second column of Table 3.2 can be considered as a measure of the robustness of a solution. Following this argument, the solution from the PH approach appears to be the most robust one.

Table 3.2. Evaluation of the solutions on 1,000 scenarios

| | Eval. vs. Benchmark$_{1,000}$ | Eval. vs. Obj. |
|---|---|---|
| Algorithm 5 | -9.8% | +11.2% |
| Algorithm 6 | -1.3% | +17.1% |
| Algorithm 7 | -22.6% | +1.6% |

**Sensitivity Analysis on Cost Coefficients.**

We perform sensitivity analysis to examine how well the proposed algorithms work when operating cost[4] increases (or decreases) by 5 or 10%. The computational results are presented in Table 3.3. All the evaluations are based on the 32 generated scenarios. As we can see, the solutions given by the proposed algorithms are still 10 - 15% better than the benchmark however the operating cost varies. The only exception happens when cost increases by 5%, in which case Algorithm 5 yields only a 1.2% improvement over the benchmark while Algorithm 7 gives a surprisingly high 23.3%. Algorithm 7 seems to dominate the other two in all cases, and from Figures 3.5 - 3.8, we can make a similar argument that it is able to reach a better solution faster. However, this dominant performance of Algorithm 7 does not imply that the two Benders decomposition-based algorithms are no longer necessary. In fact, if a longer execution time is available and an optimal solution is desired, we still need the two Benders algorithms since Algorithm 7 is only a heuristic without any convergence guaranteed.[5]

From Table 3.3, we also see that the profit of the airline is very sensitive to the operating cost fluctuations. For example, a 5% decrease of operating cost leads to an 80% increase in profit, while a 5% increase reduces the profit by more than 85%. This proves the tight margin of airlines and the importance of cutting costs by utilizing equipment and labor more efficiently.

Table 3.3. Sensitivity analysis on cost coefficients

| | Obj. vs. Benchmark$_{32}$ | | | | Obj. Changes | | | |
|---|---|---|---|---|---|---|---|---|
| Cost Coef | $\downarrow$ 10% | $\downarrow$ 5% | $\uparrow$ 5% | $\uparrow$ 10% | $\downarrow$ 10% | $\downarrow$ 5% | $\uparrow$ 5% | $\uparrow$ 10% |
| Algorithm 5 | -11.6% | -11.6% | -1.2% | -11.0% | -165.7% | -79.6% | +88.1% | +171.1% |
| Algorithm 6 | -11.2% | -12.4% | -9.8% | -10.8% | -168.1% | -83.2% | +86.9% | +172.2% |
| Algorithm 7 | -13.1% | -14.7% | -23.3% | -12.3% | -167.3% | -83.2% | +85.6% | +172.4% |

[4]Operating cost refers to the cost coefficients in (3.1).

[5]Further experiments show that Algorithms 5 and 6 would prevail after 15 hours of running time.

**Sensitivity Analysis on Number of Scenarios.**

Besides operating cost, how the number of scenarios impacts the final solution is also worth of exploring because if considering more scenarios does not give a better solution, we rather consider a simpler model with fewer scenarios to expedite the solution process. Therefore, in this section, we perform the proposed algorithms on 8 and 16 scenarios, and compare the solutions to the ones based on 32 scenarios. To see how these solutions work in reality (with high uncertainty), they are compared with respect to the expected cost based on 1,000 scenarios.

Table 3.4. Sensitivity analysis on number of scenarios

| | Eval. Changes | | |
|---|---|---|---|
| | 8 Scenarios | 16 Scenarios | 32 Scenarios |
| Algorithm 5 | +6.3% | +3.5% | 0 |
| Algorithm 6 | +1.9% | +3.5% | 0 |
| Algorithm 7 | +1.3% | +0.5% | 0 |

As indicated in Table 3.4, fewer scenarios lead to worse solutions among all three algorithms. This makes intuitive sense as the more scenarios we consider, the more likely our model is able to capture the real demand fluctuation and hence give a solution working robustly under the level of uncertainty possessed by 1,000 scenarios. An exception happens for Algorithm 6, in which case 8 scenarios perform better than 16 scenarios.

## 3.5. Summary and Future Research

In this chapter, we have proposed a two-stage stochastic fleet assignment model that considers potential demand scenarios and attractiveness-based spill and recapture effects. Most importantly, we developed three MapReduce-based solution approaches to solve each scenario-dependent subproblem in parallel to reduce the computational time. To examine the efficacy of

the modeling and algorithmic strategies, computational results using real data from a medium-size airline were presented. It would be of interest of a follow-up research to apply the proposed distributed framework to other stochastic problems, such as crew scheduling and financial portfolio management. Further computational experiments should be potentially conducted to examine if there exists an upper bound on the number of scenarios; once exceeding this value considering more scenarios would not lead to any significant changes in the final solution. And further research is also necessary to incorporate additional features such as re-fleeting, flexible flight times (departure and arrival time windows) and schedule balance into our model. From the implementation standpoint, transferring the implementation to Apache Spark is also worthwhile attempting.

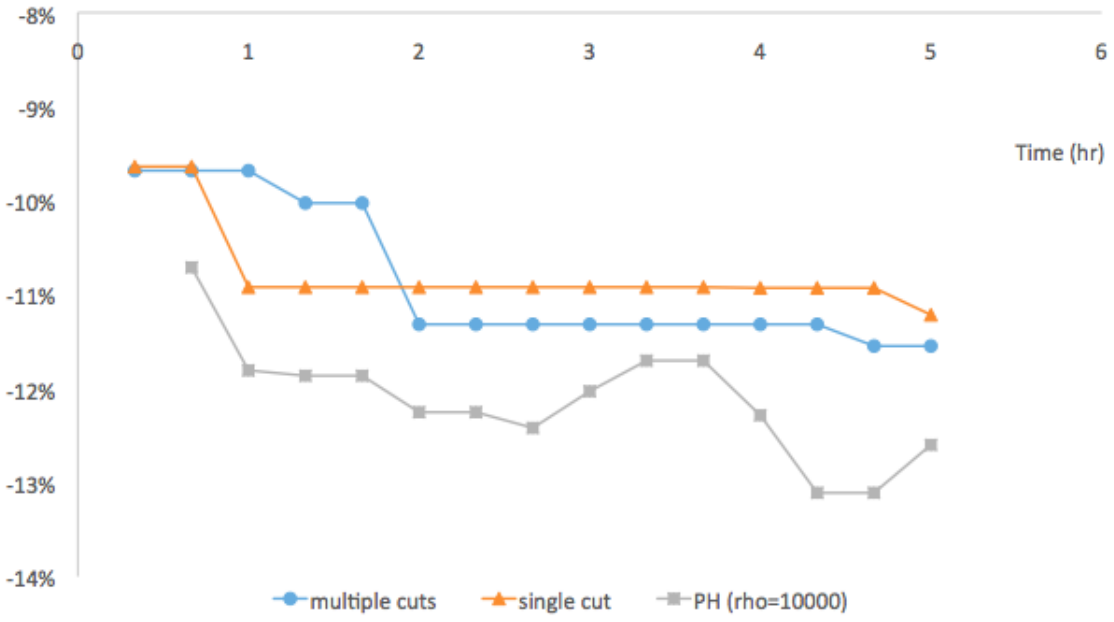


Figure 3.5. Obj. vs. Benchmark$_{32}$ in 5 hours when operating cost decreases by 10%

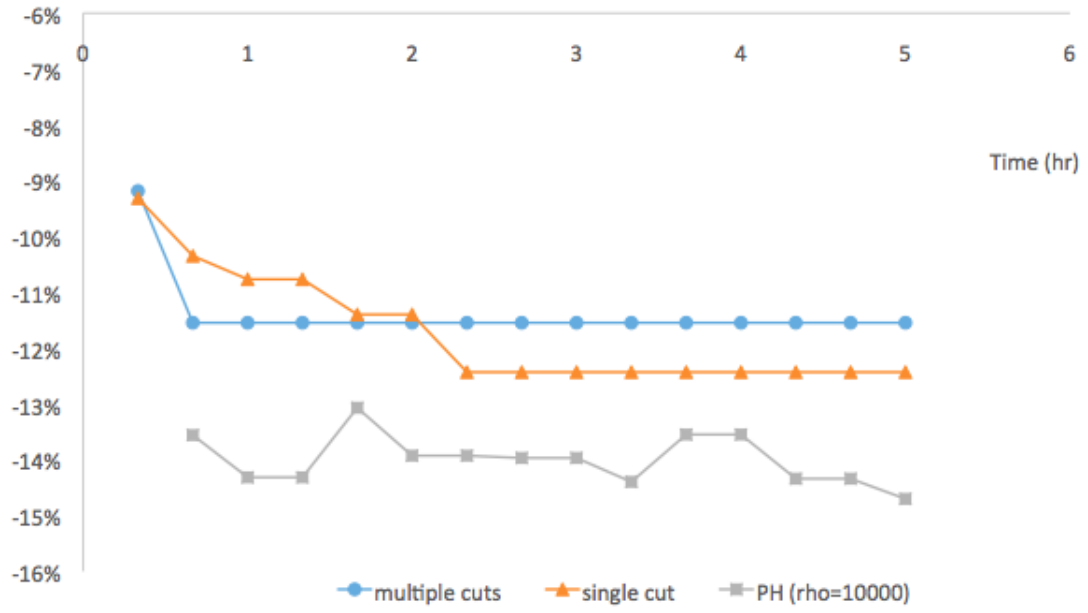


Figure 3.6. Obj. vs. Benchmark$_{32}$ in 5 hours when operating cost decreases by 5%

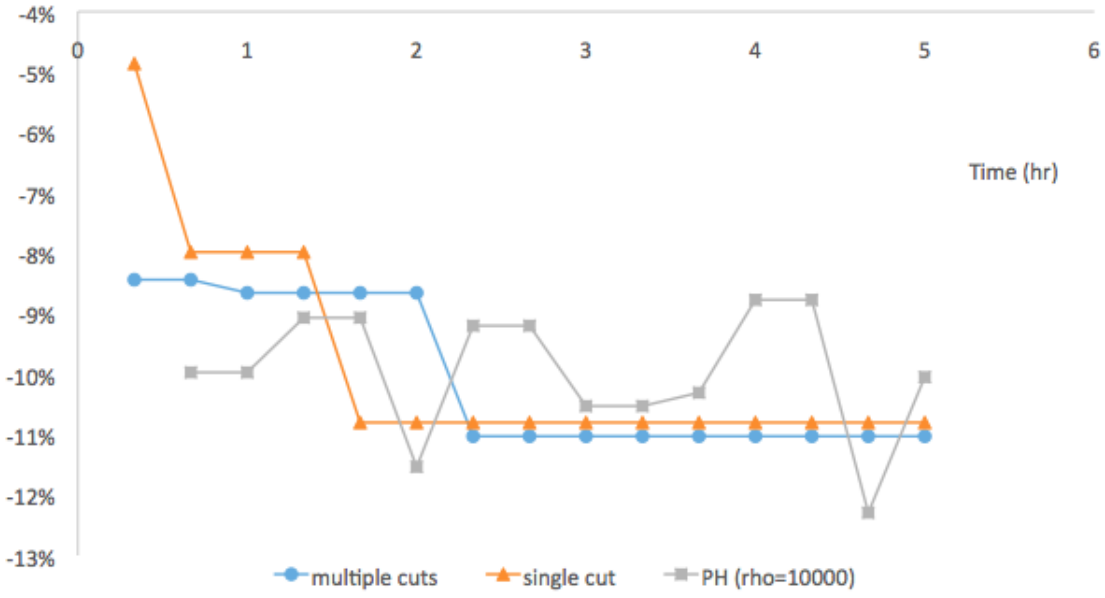


Figure 3.7. Obj. vs. Benchmark$_{32}$ in 5 hours when operating cost increases by 10%

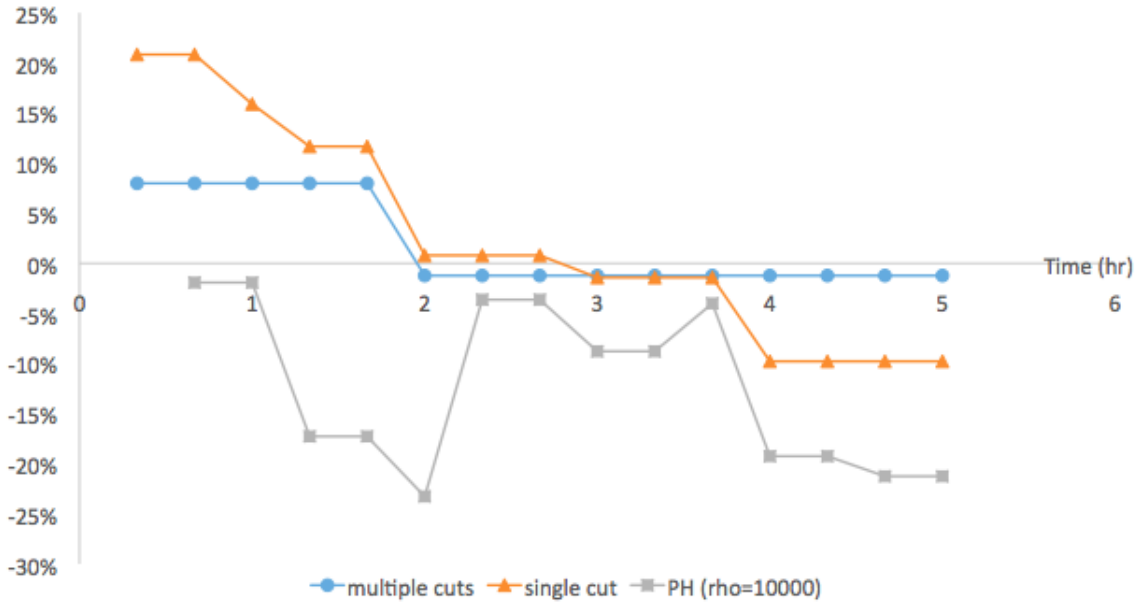


Figure 3.8. Obj. vs. Benchmark$_{32}$ in 5 hours when operating cost increases by 5%

CHAPTER 4

# Large-scale Adversarial Sports Play Retrieval with Learning to Rank

## 4.1. Introduction

As sports games are more and more analytically driven, the collection of sports tracking data has quickly become the norm. For instance, STATS LLC's SportVU basketball tracking system generates location coordinates for every player, ball, and referee 25 times per second along with detailed logs of events such as shots, passes, fouls, etc. However, given the overwhelming amount of data being collected, a large-scale information retrieval system that can effectively manage and access sports plays has not yet surfaced. To close this gap, we present a retrieval system that can quickly find and prioritize similar basketball plays from a large number of games (e.g., one season or multiple seasons). With our system, coaches and scouts no longer need to spend dozens of hours watching video footage in order to find a specific play of interest. Once an input query is issued, the system is able to return a ranked list of similar plays ordered by the relevance to the query at interactive speeds. It is by nature a search engine in the field of sports, and although we focus on basketball, it is straightforward to extend the context to other sports like soccer or American football.

An immediate question that arises from the development of such a search engine is how to design a query format that is both intuitive to use and rich enough to capture the spatiotemporal nature of basketball plays. Unlike traditional search engines (e.g., Google, Yahoo) that use keyword queries, it is almost impossible to specify basketball plays with keywords. On one

hand, if we use simple keywords like “three-point shot,” the number of retrieved plays can easily reach several thousand or more. Users still have to browse through a large collection of results before finding the specific targets. On the other hand, if we try to describe a play in more detail - for instance, how each player and the basketball are moving and where and when the three-point shot is made - using keywords would be very tedious. Therefore, in this chapter, we employ a user interface (UI) that allows users to issue an input query either by selecting a play through browsing previous games or by drawing a play of interest on an interactive mobile device similar to how coaches draw plays on white board.

Another challenge is the vast amount of data the system has to store and process in real time. For example, SportVU generates over 100GB of tracking data for a season of games, and building a database of multi-agent trajectories with each trajectory consisting of hundreds of tracking records can easily reach several terabytes. In this chapter, a trajectory is defined as a spatial trait of a player or ball over a short period of time, and a play is simply a collection of trajectories of one or more players and the ball. Our system is designed to handle multiple seasons through a carefully designed database model on top of key-value databases. To get around the scalability challenge, we propose an encoded representation of basketball plays based on k-means clustering and player’s role alignment. In particular, we perform k-means clustering separately on ball, offensive player and defensive player trajectories and replace each trajectory in a play with the index value of the closest centroid. This representation dramatically reduces the amount of data needed to be stored and also enables fast identification of candidate plays. A *key-value* database using the ball trajectory index as a primary search key is constructed to store the encoded plays, and a streaming-based distributed framework is developed to process candidate plays at scale to achieve real time performance. Given an input query, we first compare its

ball trajectory against the centroids of ball trajectories and acquire all the plays that resemble the query at least from the ball trajectory perspective. We then compare each candidate play to the query in parallel and decide the top candidates to be returned to the user.

Finally, a good search engine would personalize search ranking by mining each user's search log and clickthrough behavior. This is generally known as learning to rank in information retrieval. Similar functionality is achieved on our system through a pairwise ranking approach - rankSVM. An advanced feature representation of basketball plays learnt via a convolutional neural network (CNN)-based autoencoder is used in training the ranking model.

In the numerical section, we show that the response time of our system is around 5-7 seconds once an input query is issued. Considering the granularity of our data and the (extremely large) number of plays being processed, it is a remarkable speed according to domain experts' opinions. We demonstrate the effectiveness of our learning to rank approach by evaluating the predicted rankings using real user feedback. Experiments on how to learn the encoded representation are also presented.

The contributions of this work are as follows.

(1) It is the debut of the first information retrieval system (search engine) in the domain of sports that can process industry-sized data. The concept and algorithms can be easily extended to a variety of other sports such as soccer, ice hockey and American football.

(2) According to the authors' best knowledge, this is the first time that multi-agent sports plays are encoded using clustering-based indices (obtained through clustering on both player and ball trajectories). The associated key-value database leads to a promising direction of accessing and managing overwhelmingly large sports tracking data.

(3) This is also the first effort to apply learning to rank to sports play retrieval. It makes our system more adaptive to user's specific interest and significantly improves rank quality.

(4) Finally, according to authors' best knowledge, we are the first to use convolutional autoencoder to extract features from sports tracking data. The proposed CNN model unveils the dynamic, hidden structure of basketball plays and could potentially be applied to analyze, predict and simulate sports games.

The rest of this chapter is organized as follows. In Section 4.2, we discuss the related literature. Section 4.3 illustrates the major components of the system, and Section 4.4 gives implementation details and the findings from our computational experiments. Finally, we conclude in Section 4.5.

## 4.2. Related Literature

Previous work on information access in the sports domain largely focuses on improving categorization of plays [52, 18, 10]. Sha et al. [58] develop the first sports play retrieval system that allows users to issue an input query by drawing a play of interest. Through alignment, templating and hashing techniques tailored for multi-agent trajectories, the system is able to search the most relevant plays from a few dozen games. Although our system employs a similar query paradigm that allows users to either select or draw a play, the core parts are completely different. With a key-value database that stores the encoded plays and a distributed architecture, we are able to retrieve from multiple seasons at interactive speeds. Sha et al. [58] use only ball trajectory clustering, while we also perform clustering on player trajectories to cope with scalability. Furthermore, we apply deep learning techniques to obtain an advanced similarity

metric, and use it in training a user-specific ranking model. This is definitely not addressed in [58].

There are plenty of previous works studying how to measure distance/similarity between trajectories [69, 79, 17, 80, 22], but the majority of them focus on comparing single trajectories rather than multi-agent ones. In our learning to rank approach, we compare two plays, each one consisting of 11 player and ball trajectories, based on features learnt through a CNN-based autoencoder. While in literature neural networks have been widely used in sports prediction [32, 47, 50, 75] or classification [72, 6], we have not yet seen it used for measuring similarity of sports plays. The features revealed by the convolutional neural network preserve more information than the Euclidean distance based on role alignment [11, 74, 48], and thus reveal more of the structural essence of basketball plays.

Modern search engines all utilize clickthrough data to improve rank quality. Recent works by Joachims et al. [37, 38, 39] and others [4, 3] have shown the value of incorporating implicit user feedback into the ranking process. Learning to rank is widely employed in a variety of applications in information retrieval, natural language processing and data mining [43, 46]. Solution methodologies include pointwise approaches [21, 44, 59], pairwise approaches [37, 27, 12, 14], and listwise approaches [15, 77, 78, 67]. In this study, we follow a pairwise approach, namely rankSVM, to train a linear classifier using both the features from our CNN-based autoencoder and self-crafted features.

### 4.3. A Search Engine of Basketball Plays

In this study, we use over 1,100 games of tracking data from the 2012-2013 and 2015-2016 NBA seasons. The data was collected by STATS LLC's SportVU tracking system, and contains

location coordinates for every player, ball and referee along with detailed log information such as team and player identifications (ID), time, and events. To develop a search engine that can effectively retrieve from this large number of games, we first extract all the plays from the tracking data. A play is defined as trajectories of one or more players and the ball that are organized to achieve a certain outcome (e.g., a shot made or free throw attempt) within a short period of time. Similar to [58], in this study, we only consider plays with time-windows of 1, 2, 3, 4, and 5 seconds. We then encode each play with 11 indices based on k-means clustering and player's role alignment. A key-value database is constructed to group the encoded plays based on ball trajectory similarity. That being said, once an input query is issued, we can quickly identify all the candidate plays that possess similar ball trajectories to the query. With a distributed framework, each candidate is compared to the query in parallel and once the top results are determined, we apply a user-specific rank function to them before displaying to users. The user-specific rank function is learnt via a pairwise learning to rank approach and an advanced similarity metric leveraging deep learning techniques.

In this section, we first propose the encoded representation and a key-value database to store the encoded plays. A streaming-based retrieval process built upon a distributed framework is then elaborated. Finally, we discuss how to learn a similarity metric via a CNN-based autoencoder and how to train a user-specific rank function based on it.

### 4.3.1. An Encoded Representation of Basketball Plays

We first extract all the plays with time-windows of 1, 2, 3, 4, and 5 seconds from the tracking data. To do this, we use each second as the starting point and get the multi-agent trajectories for various time-windows. This gives us over 15 million plays (over 3 million for each

time-window) for a season of games, and an unnecessarily large database to store them as consecutive location coordinates.[1] Therefore, the following encoded representation is proposed to dramatically reduce the amount of data needed to be saved and facilitate fast access to the potential candidates. The first step is to match the player trajectories of each play to 10 unified roles (i.e., offensive PG, SG, SF, PF and C[2]; defensive PG, SG, SF, PF and C) using the role alignment algorithm proposed in [11, 74]. This role-based representation reveals the intrinsic nature of basketball plays regardless of players' identities and positions, and thus leads to more accurate retrieval. Figure 4.1 illustrates the intuition behind the role-based representation. In 4.1(A), Kyrie Irving (PG) and Kevin Love (PF) are running classic pick-and-roll and in 4.1(B), LeBron James (SF) and Tristan Thompson (C) are running an almost identical play. If we align the trajectories based on player's identity or position, they would be deemed as very different plays. However, if we assign both Kyrie and LeBron to the offensive point guard role and Love and Thompson to the offensive power forward role, the two plays would be considered similar.

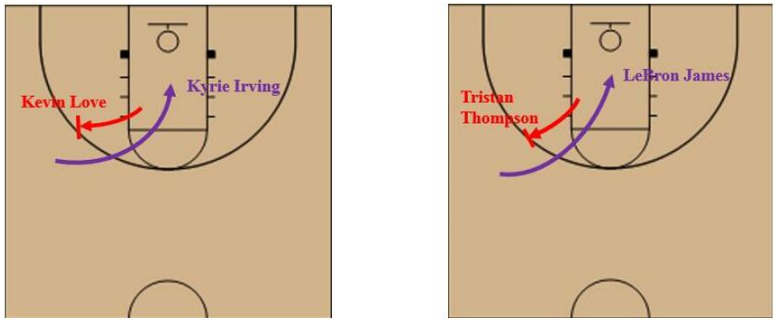


Figure 4.1. (A) Kyrie Irving and Kevin Love are running pick-and-roll; (B) LeBron James and Tristan Thompson are executing an almost identical play.

[1]Each tracking record is saved multiple times. For example, two consecutive 5-second plays have 80% overlap in terms of $(x, y)$ coordinates, and a 3-second play completely overlaps with the 5-second one starting at the same time frame.

[2]Basketball terminology: PG - Point Guard; SG - Shooting Guard; SF - Small Forward; PF - Power Forward; C - Center.

Once aligning the trajectories in the order of offensive PG, SG, ..., defensive PG, SG, ... and the ball, we normalize a play so that the offensive team always attacks the same basket. This ensures that both plays happening on the same side of the court and plays on opposite sides are compared consistently. Finally, we perform k-means clustering based on frame-by-frame $L_2$ distance separately on ball, offensive player and defensive player trajectories for various time-windows and encode each play based on the clustering results as follows. Let $C_b^t$, $C_o^t$ and $C_d^t$ be the respective clustering results for time-window $t$. To encode a $t$-second play, we first compare its ball trajectory to the centroids in $C_b^t$ and replace it with the index value of the closest centroid. We then do the same to player trajectories by comparing the offensive player trajectories against the centroid of every cluster in $C_o^t$ and comparing the defensive player trajectories against those in $C_d^t$, and obtaining the best match. This results in 11 indices for each play. As we can see, this encoded representation does not provide direct restoration to the original trajectories as it is based on centroids, but in the retrieval process, we do find them from the database exhibited next.

### 4.3.2. Database Model

We propose a key-value database model that consists of two tables. In such databases, a table is a list of nested key-value pairs, where each data record is indexed through a primary search key and a number of secondary keys. In the first table, denoted as the *playbook*, we store plays of various time-windows (1 - 5 seconds) based on the encoded representation. In particular, we group all the plays that have the same ball trajectory index and time-window using the corresponding (compound) primary search key. With this setup, given an input query of $t$ seconds,

we can quickly find all the plays that resemble the query at least from the ball trajectory perspective, and thus reduce the number of candidates to a manageable level for further comparison. This also makes intuitive sense as in basketball all players' movements are centered around the ball. If ball trajectories of two plays significantly differ (e.g., a three-point shot and a fast-break layup), the players' trajectories can hardly be similar.

For proper indexing, secondary keys are chosen to be season, game and quarter IDs, and starting and ending time of the play. These allow us to locate any exact play from historical games. Each play is stored in its encoded form using the 11 indices. To allow richer queries featuring a specific event (e.g., offensive rebound, assist) or between particular teams, a list of major events associated with the play and the offensive and defensive team IDs are also included. The structure of the table is depicted in Figure 4.2.

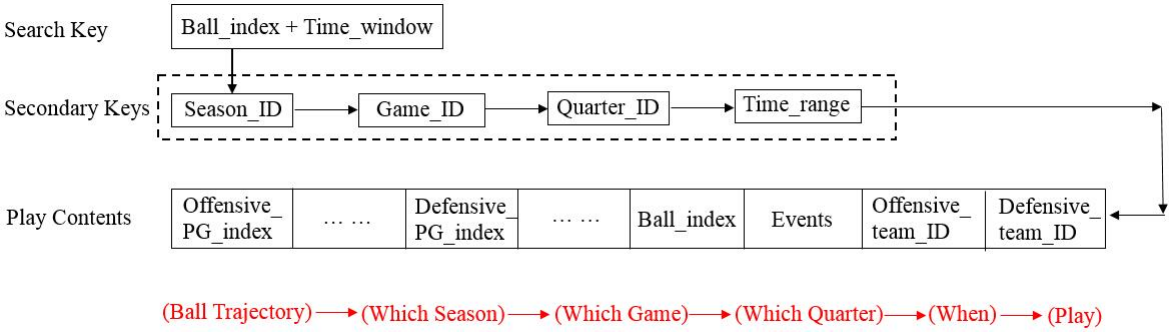


Figure 4.2. Structural flow of the playbook

We have another table that stores tracking data in its original format. We call it the *trackbook*. As shown in Figure 4.3, the search key is simply the game ID and the secondary keys are quarter ID, in-game time and record ID. In-game time allows us to fetch tracking data at the frame level, and record ID is used to differentiate each player and the ball at any given time. Each tracking record consists of the location coordinate along with other information such as

the player ID, role and event. This table jointly works with the playbook to restore any encoded play back to the original trajectories, which plays a crucial role during the retrieval.

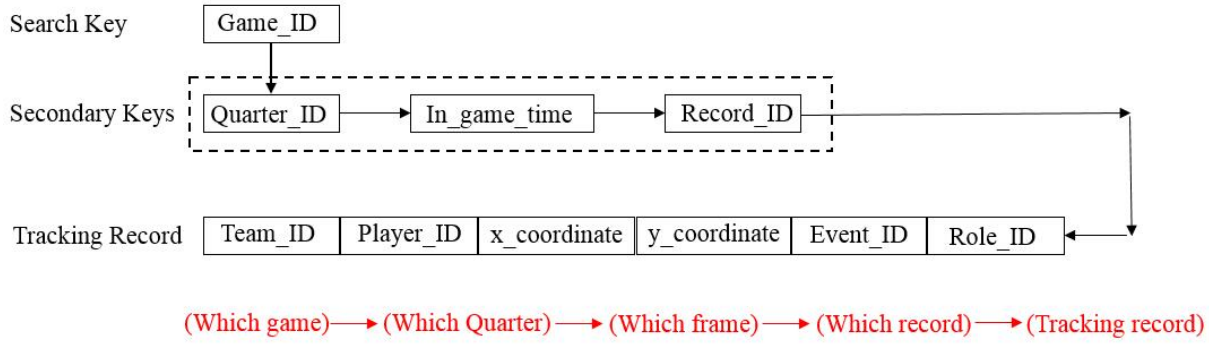


Figure 4.3. Structural flow of the trackbook

### 4.3.3. Distributed Query-Candidate Processing

As shown in Figure 4.4, the overall retrieval process can be summarized as follows. Given an input query of $t$ seconds, we first compare its ball trajectory against the centroids of every cluster in $C_b^t$ and find the closest match. This gives us the primary search key in playbook. After constructing all secondary keys (if any are present in the query), we next search the playbook and fetch a list of candidate plays that have ball trajectories similar to the query's. Notice that at this point, the candidate plays are encoded with 11 indices. To determine which candidates are most relevant to the input query, the encoded representation is converted back to trajectories using centroids. For instance, if a candidate play has ball trajectory index $n$, $n = 1, 2, ...,$ we use the $n$th centroid of $C_b^t$ to approximate its ball trajectory. By doing the same to the other player indices, we re-establish (approximate) each candidate play with 11 centroid trajectories. Since our approach aligns trajectories based on player's role, we can compare these

re-established candidate plays to the query play by calculating $L_2$ distance between trajectories that have the same role. The 11 distances can then be summed up as a single similarity measure. We choose $L_2$ instead of other more sophisticated metrics such as dynamic time wrapping or longest common subsequence because: 1) it is easy to use; 2) according to [58], using other metrics does not lead to any significant improvement in this step.

Each candidate play is re-established and compared to the query in parallel. Once the top candidates and the ranking are decided, we use information contained in the secondary keys to obtain original tracking data of each top result from the trackbook and generate output accordingly. This is how we restore to the original trajectories as mentioned in Section 4.3.1. As noted in Figure 4.4, the comparison and ranking are based on approximate plays while only the real plays are shown to users. The question mark represents a user-specific rank function, which is to be elaborated next.

### 4.3.4. Learning to Rank for Sports Play Retrieval

The ranking obtained so far is based on agent-to-agent $L_2$ distance. However, given a user's search and clickthrough history, an adequate search engine would be able to capture user-specific interest and adjust the search ranking accordingly. To this end, a pairwise learning to rank approach based on rankSVM is taken for learning a user-specific rank function based on the clickthrough data and the similarity metric learnt via a convolutional autoencoder. Let $Q$ be the set of all queries as part of the training data set. For each query $q \in Q$, we have ordering $(v_1, v_2, v_3, ...)$ of the returned results from the retrieval system and sequence $C^q$ containing the ranks of the clicked-on results in the order of clicking. The goal is to re-order $(v_1, v_2, v_3, ...)$ so that it reflects the relevance judgments conveyed by $C^q$.

To obtain an embedding capturing more structural differences between basketball plays, we apply CNN to the 2D visualizations of trajectories. Based on [32], we produce visualization images by breaking the 94 by 50 feet court down to 1 square foot regions corresponding to individual pixels. In order to identify offense and defense, we use 3 channels to separate offensive players, defensive players and the ball. For each player and ball trajectory, and at each time frame $t$, we place a 1 at the pixel location of the corresponding channel that contains $(x_t, y_t)$. The CNN-based autoencoder takes these $3 \times 94 \times 50$ visualization images from each play as input. The tailored network configuration is specified in Section 4.4.1. The output of the last encoding layer is a compressed feature representation. These features are then used to describe the match between query $q$ and result $v$ as follows.

$$\Phi(q,v) = [f_v,\ f_v \circ f_q,\ d_{qv}^1, ..., d_{qv}^t],$$

where $f_q$ and $f_v$ are feature vectors from the autoencoder, and $d_{qv}^i$, $1 \leq i \leq t$, is a list of role-to-role $L_2$ distances during the $i^{th}$ second. The time-window $t$ can be inferred from either $q$ or $v$, and $f_v \circ f_q$ is an element-wise product (Hadamard product). We include this product term to preserve information of $q$. We use $f_v \circ f_q$ instead of just $f_q$ because rankSVM is trained on pairwise subtractions of $\Phi$'s and hence using $f_q$ would result in all zeros and complete loss of information regarding $q$.

Next, we extract users' preference feedback from the clickthrough data. Since based on [37, 38], the clickthrough data does not convey absolute relevance judgments, but partial relative relevance judgments for the clicked results, in this study, we extract only pairwise preferences. Given $C^q$ for query $q$, we extract these pairs from a display page based on: 1) clicked results are more relevant than unclicked ones; 2) prior clicked results are more relevant than later clicked

ones. Further details are provided in Section 4.4.1. A linear rankSVM classifier $\vec{w}$ is then trained based on $v_i <_{r^*} v_j \Leftrightarrow \vec{w}\Phi(q, v_i) < \vec{w}\Phi(q, v_j)$, where $<_{r^*}$ denotes the user preferred ranking, using the stochastic gradient descent (SGD) method. User-specific ranking based on a test query can hence be achieved by applying the resulting $\vec{w}$ to the pairwise combinations of the top results.

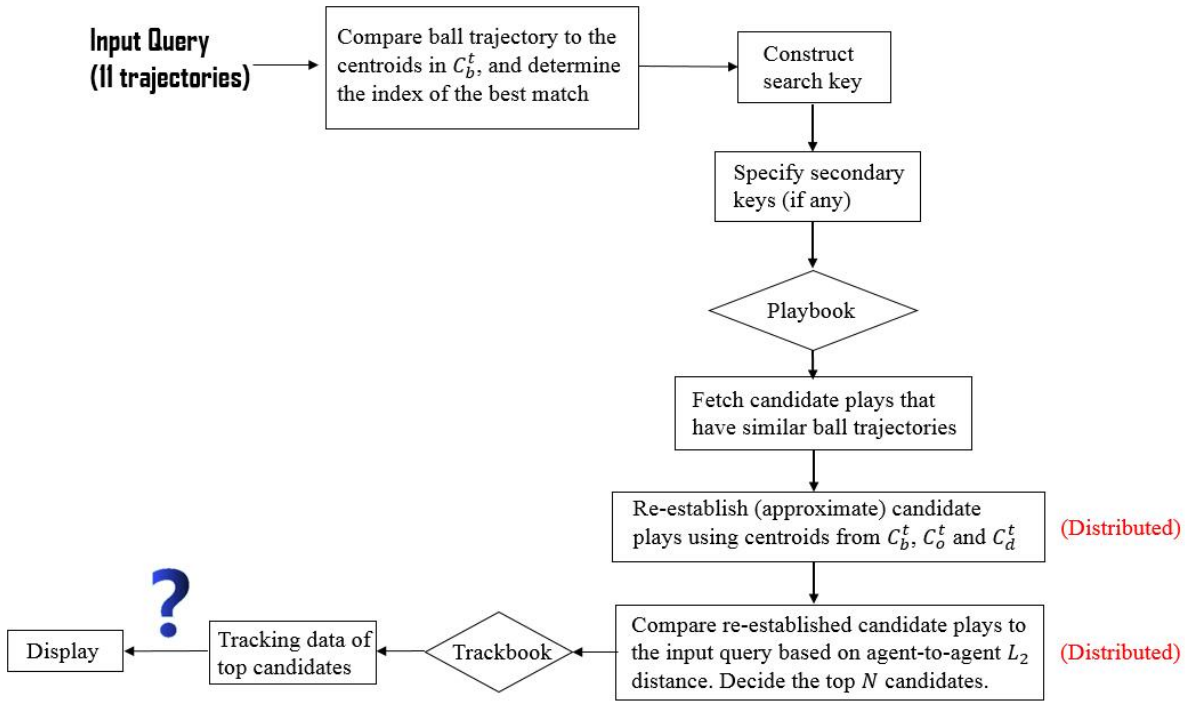


Figure 4.4. Summary of the streaming-based retrieval process

## 4.4. Implementation

### 4.4.1. Implementation of the Retrieval System

The retrieval system is executed on Amazon Web Service Elastic Compute Cloud. The cluster consists of 8 m3.xlarge instances, each having 4 cores and 15GBytes of memory. Apache

Spark version 1.6.0 is the backbone engine for distributed data processing. The streaming-based retrieval is achieved via Spark Streaming, a Spark generic API for batch processing. We chose Cassandra 2.1.5 as our key-value database because: 1) it is open-source; 2) it scales up easily for future seasons; and 3) it excels with Spark.

The data was generated by STATS LLC's SportVU tracking system. It utilizes 6 cameras to track all player locations (including referees and the ball) 25 times per second along with detailed log information such as pass, shot, dribble and turnover. This dataset is unique since before SportVU, there was very little data available to track player movements and most of the sports analysis was performed based on elementary statistics such as points and rebounds. We use over 1,100 games of tracking data from the 2012-2013 and 2015-2016 NBA seasons in this work.

We develop the following user interface that allows users to issue an input query either by selecting a play from previous games or by drawing a play of interest similar to how coaches draw plays on boards. Figure 4.5 shows how users can select a specific play while streaming a game, and Figure 4.6 gives an example of how to draw a play using the drawing tools. It is similar to [58], but instead of browsing the visualizations of game trajectories, users can watch real games via our interface.

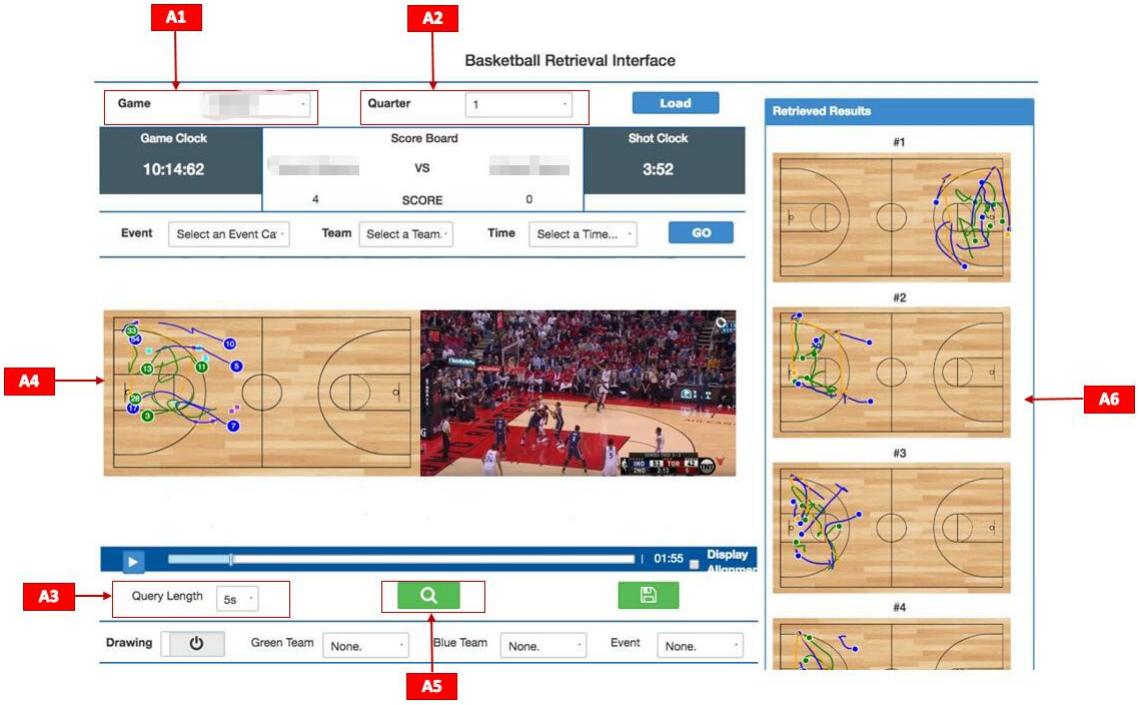


Figure 4.5. On the browsing interface, users can (A1) select a game, (A2) select a quarter, (A3) specify the time-window of the play and (A4) start watching the game (right) and the corresponding 2D visualization (left). Once a play of interest is found, users can (A5) click the retrieval button and (A6) the system returns all the similar plays in the display panel. The ball trajectory is in yellow, and the offensive and defensive player trajectories are in blue and green respectively.

We perform k-means clustering separately on ball, offensive player and defensive player trajectories for various time-windows (1-5 seconds). Due to the volume of the data, the clustering is done in Spark. We choose the number of clusters $k$ so that each cluster in $C_b^t$ contains less than 1,000 trajectories, and each cluster in $C_o^t$ and $C_d^t$ contains less than 5,000 trajectories. To ensure this, we keep dividing clusters that have more than 1,000 (or 5,000) trajectories into sub-clusters by applying another round of k-means until the criterion is met.

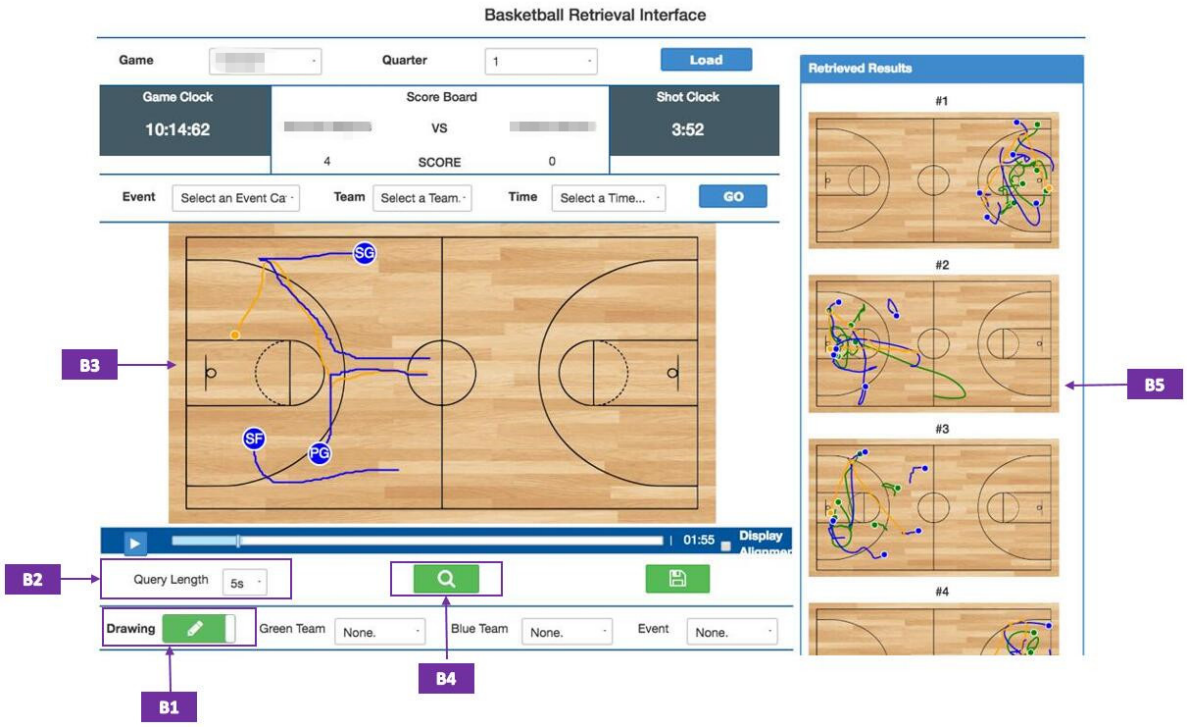


Figure 4.6. On the drawing interface, users can (B1) select the drawing tool, (B2) specify the time-window of the play and (B3) draw a play of interest. Users can then (B4) click the retrieval button and (B5) the system returns all the similar plays in the display panel.

The retrieval system comprises of a frontend interface and a Spark streaming-based backend engine. Once an input query is issued via the interface, either by selecting from a previous game or by drawing, a comma separated values (csv) file that contains the query information is generated and pushed to a directory in S3[3]. The backend engine Spark streaming monitors the corresponding directory and processes any files created there. With this setup, we avoid going through the Spark-submit initialization steps every time a new query is issued, and thus significantly reduce the response time. This is definitely not the case if each query is submitted

[3]S3 stands for Amazon Simple Storage service.

to the backend engine as an individual application. Finally, the retrieval output is also saved to S3 based on the final ranking, and the frontend simply plots and displays them sequentially.

Through extensive numerical experiments, we finalize the configuration of our CNN-based autoencoder as 7 convolutional layers and 2 max-pooling layers in the encoding phase, and 7 deconvolutional layers and 2 up-sampling layers in the decoder. We use the typical KL-divergence as the loss function, and ReLU (rectified linear unit) as the activation function. Adaptive subgradient method is utilized to minimize the loss function during training. The whole network is implemented in Keras with Theano backend, and the parameters are calibrated based on validation loss and the quality of the reconstructed output. The inputs to the autoencoder are the $3 \times 94 \times 50$ images described in Section 4.3.4, and the embedding has dimensionality $8 \times 10 \times 5$.

Lastly, based on the eyetracking experiments by Joachims et al. [38] and the unique design of our interface[4], we propose the following ways to interpret users' clickthrough behavior, which generalize the single-page logic described in Section 4.3.4.

1) For each pair of clicked results $(v_i, v_j)$ on the same page, we have $v_j <_{r^*} v_i$ if $v_j$ is ranked lower but clicked before $v_i$.
2) For each pair of results $(v_i, v_j)$ on the same page, we have $v_j <_{r^*} v_i$ if $v_j$ is clicked but $v_i$ is not.
3) For each pair of results $(v_i, v_j)$ with $v_i$ on a higher ranked page and $v_j$ on a lower ranked page, we have $v_j <_{r^*} v_i$ if $v_j$ is clicked but $v_i$ is not.

Consider the following example. A user issues an input query, and then clicks results 5, 1, 3, 4 on the first page, and result 8 on the second page. Since the fifth result is clicked before results 1, 3 and 4, according to 1), we have $v_5 <_{r^*} v_1$, $v_5 <_{r^*} v_3$, and $v_5 <_{r^*} v_4$. Analogously, 2) implies

[4]We display only 5 results on each page. Users are supposed to make eye contact with all 5 results before making any clicks.

that $v_1 <_{r^*} v_2$, $v_3 <_{r^*} v_2$, $v_4 <_{r^*} v_2$, $v_5 <_{r^*} v_2$ on the first page, and $v_8 <_{r^*} v_6$, $v_8 <_{r^*} v_7$, $v_8 <_{r^*} v_9$, $v_8 <_{r^*} v_{10}$ on the second page. Finally, 3) gives us $v_8 <_{r^*} v_2$. The rankSVM model is trained using simulated data (described next) and solved by the SGD algorithm implemented in Python scikit-learn.

### 4.4.2. Simulation of the Clickthrough Data

As mentioned, the rankSVM model is trained on simulated data because:

(1) Our sports play retrieval system is the first of its kind and hence there is no historical data to exploit.

(2) The target user base is relatively narrow and the frequency of usage is also much lower than in typical search engines. Therefore, it is almost impossible to collect extensive user feedback in a short period of time.

(3) The model based on simulated data leads to rankings that well capture user-specific interests expressed in the test data set (explained later).

We simulated a variety of user preferences including: users focusing on close shots (15%), users focusing on mid-range shots (5%), users focusing on three-point shots (25%), users focusing on shot attempts from the corner (5%), users focusing on shot attempts from the wing (5%), users focusing on shot attempts from the top (5%)[5] and normal users focusing on ball trajectory similarity (40%). The percentages are designed to reflect the league's recent trend in three-pointers and close shots (dribble layups, dunks, and free throws). All the input queries are randomly generated from previous shot attempts and the clickthrough behavior is simulated based on the following.

[5]Corner shots: shot attempts with shot angle $0-30^\circ$ on both sides of the court; wing shots: shot attempts with shot angle $30-60^\circ$ on both sides of the court; and top shots: shot attempts on top of the three-point line.

(1) We start from the **first page**: for users that have a specific interest, if there is any result (out of 5) matching the preference, click all the qualified results based on their original ranking.

(2) For all users, if the ball trajectory in any result is very close to the query's (i.e., $L_2$ distance per frame is less than 3.5 feet), click all the qualified results based on their original ranking.

(3) All the remaining results are clicked according to a base probability and how close the ball trajectory is to the one in the query. In particular, the probability for each result to be clicked is: base $\times \left(L_b^*/L_b\right)$, where $L_b$ denotes the $L_2$ distance between the query and result's ball trajectories and $L_b^*$ is the smallest $L_b$ among all the returned results. The base probabilities are set as 0.9, 0.9, 0.85, 0.8 and 0.75 for the 5 results on the first page.

(4) We then proceed to the **second page**. It has 0.6 probability to be browsed, and if so, repeat steps 1 - 3 with base probability 0.5 for all 5 results.

(5) Repeat step 4 for the **third page**, if any, with 0.15 probability to be browsed and base value 0.25.

(6) Repeat step 4 for the **fourth page**, if any, with 0.05 probability to be browsed and base value 0.2.

All the probabilities are specified based on the eyetracking experiments in [38]. We assume no more than 20 results are returned for each query and simulate 22,000 search sessions with time-window 5 seconds. The total running time of the simulation is 7 days (one week), which dictates the number of search queries. We split 5% as the validation set, and leave the rest as the training set.

## 4.5. Numerical Results

We validate our system via a user study. All participants are basketball domain experts who have previous experience with sports data analytics and naturally understand the concept of our search engine. They were asked to issue queries either by selecting from a previous game or by drawing a play of particular interest, and scan the results top-down and click retrieved plays that they think are most relevant to the input query. The average response time of our system to display the first result is around 5-7 seconds, and the overall processing time to display all 20 (default value) results is less than 15 seconds. All participants prefer to use our system for basketball play retrieval over the traditional approaches (e.g., watching video footage) and are satisfied with the response time. They were also asked if they think the system has done a "good" job in finding the most similar plays for each query, and the percentages of a "yes" answer are listed in Table 4.1. As the numbers suggest, around 90% of the time, the participants are satisfied with the retrieval quality.

Table 4.1. Satisfaction rate of the retrieval quality

| | Participant 1 | Participant 2 | Participant 3 |
|---|---|---|---|
| yes/no | 35/5 | 37/3 | 27/3 |
| Satisfaction rate | 87.5% | 92.5% | 90.0% |

By examining the queries issued by the participants, we also verify that our system performs equally well on common queries (e.g., dribble and pass) and on relatively rare queries (e.g., fast break). As shown in Figures 4.7-4.9, the system not only retrieves well for a normal penetration play (Figure 4.7), but also returns high quality results for plays like fast break (Figure 4.8) and offensive rebound (Figure 4.9). Note that in these images, red depicts ball, blue offensive players and green defensive players.

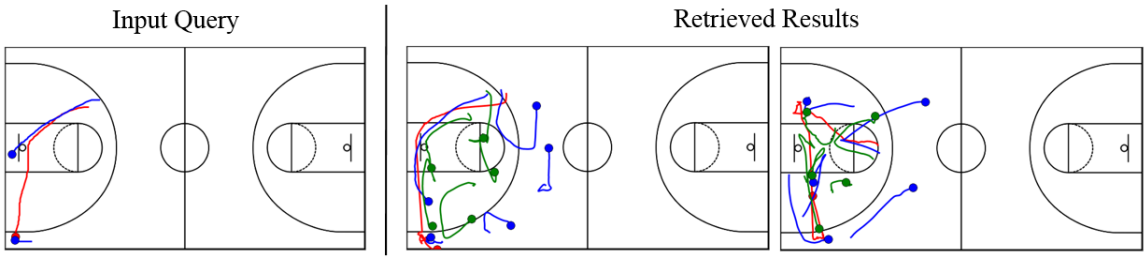


Figure 4.7. Selected retrieval results for a penetration and pass

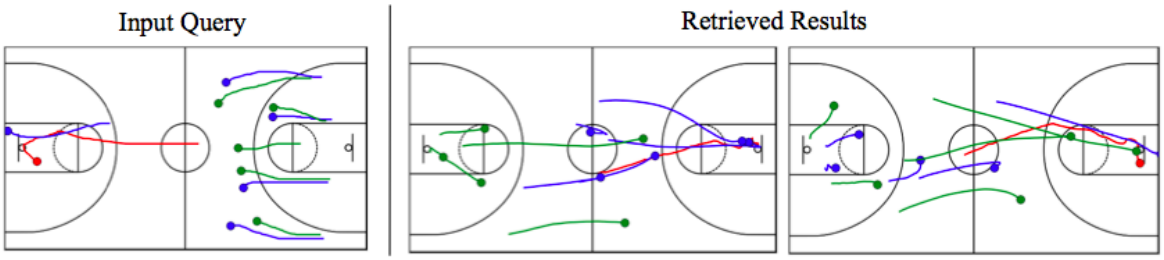


Figure 4.8. Selected retrieval results for a fast break layup

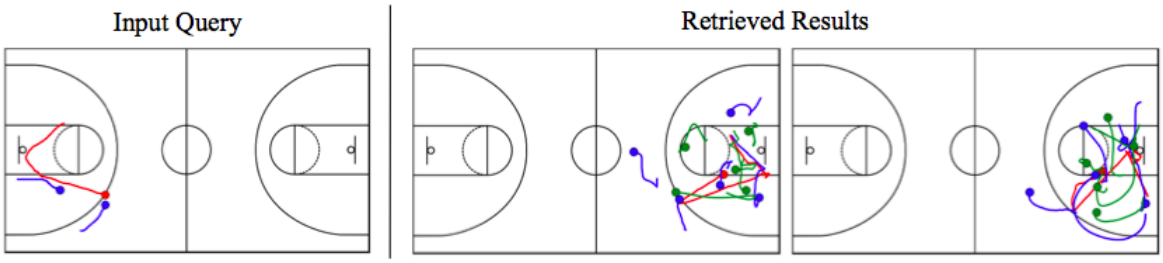


Figure 4.9. Selected retrieval results for an offensive rebound and pass out

Next, we show the effectiveness of learning to rank by evaluating the predicted ranking using participants' clickthrough data. The evaluation is based on metrics that are widely used in information retrieval: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (nDCG) and Kendall's $\tau$. We omit the details of these metrics since they have been well documented in a variety of literature [31, 37, 51, 40] and are out of the scope of this chapter. We tune the parameters of the rankSVM model (e.g., number of epochs and $L_2$ regularization

multiplier) by a grid search and assessing MAP on the validation data. Due to the specific design of our display interface (i.e., displaying 5 results on each page), we have three ways of applying the rank function: 1) apply to all the top results; 2) sequentially apply to the results on each page and always rank those on previous pages higher; 3) sequentially apply to the results on each two consecutive pages[6]. Therefore, we repeat tuning three times to get the best model for each measurement setting.

We apply the three models to the 110 test queries from our user study. The predicted rankings are then evaluated based on MAP, Kendall's $\tau$ and nDCG at $K$. The results are reported in Table 4.2 and Figure 4.10, and all the values are averaged across the test queries. As we can see, the MAP values are all on the high-end especially when we apply the model sequentially on each page or on every 2 pages. We can draw similar insights from nDCG at $K$. As shown in Figure 4.10, the gain is higher at each $K$ when we rank the results on each page separately. It increases from 0.702 to 0.887, suggesting really high quality rankings. Similar to MAP's, the nDCG values from the every-2-page approach indicate good rankings as well. The only exception happens to be the Kendall's $\tau$ as we achieve the highest score when applying the rank function to all the results. This is because Kendall's $\tau$ captures relationships that pagewise strategies cannot. A simple example is as follows. If result 12 on the third page is clicked while result 3 on the first page is not, then based on the criteria illustrated in Section 4.4.1, we should at least have $v_{12} <_{r^*} v_3$. However, this would be missed by either of the pagewise strategies. Fortunately, the Kendall's $\tau$ from the every-2-page approach is very close to its maximum value since users are unlikely to browse many plays from the third page onward. Therefore, based on all three metrics, applying user-specific rank function seqentially on every two pages always

[6]This makes intuitive sense as we only display 5 results on each page, and thus users are likely to proceed to the second page to examine more results after browsing the first one.

leads to good rankings and is thus recommended. Due to the fact that all the depicted numbers are higher than or at least comparable to those in the literature [3, 14], we successfully demonstrate the effectiveness of our learning rank approach with CNN-based embeddings.

Table 4.2. MAP and Kendall's $\tau$ for all strategies

| | MAP | Kendall's $\tau$ |
|---|---|---|
| Rank on each page | **0.817** | 0.158 |
| Rank on every 2 pages | 0.739 | 0.382 |
| Rank on all the results | 0.506 | **0.403** |

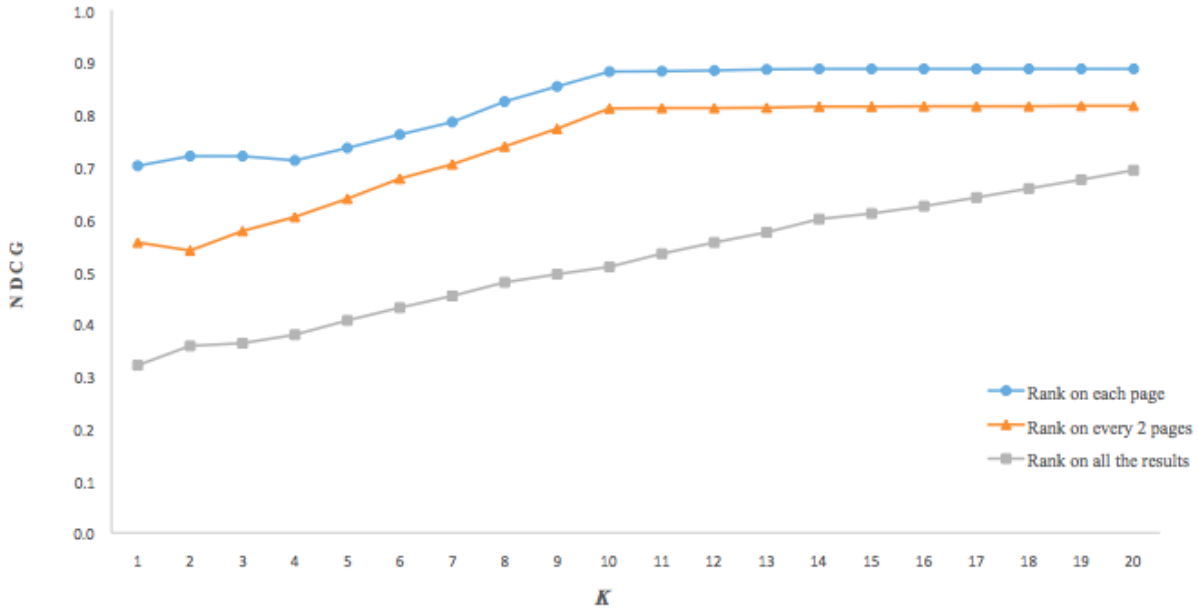


Figure 4.10. nDCG at $K$ for all strategies

We also explore if it is possible to learn the encoded representation with less data. As discussed in Section 4.3.1, the centroids are learnt from over 1,000 games and more than 3 million

trajectories. To see if we really need that much data to learn representative movement patterns, we conduct k-means clustering on monthly data of the 2012-2013 season and examine the reconstruction error from assigning a month's data to other months' clusters. In particular, the reconstruction error is computed as the sum of $L_2$ distances from any trajectory to its assigned center. We still choose $k$ so that each cluster contains less than 1,000 ball trajectories or 5,000 player trajectories. As we can see from Figure 4.11[7], for 5-second ball trajectories, the smallest errors are always achieved when assigning the data to its original cluster; however, the numbers are all close, implying that the cluster centers remain stable from each month to the full season. Larger errors are observed when assigning the whole season's data to clusters of each month, but the differences are not significant. This makes intuitive sense as the less data we use to do clustering, the fewer clusters we can get (based on how $k$ is determined) and hence the less likely our clustering result is able to capture all the representative patterns of a larger data set and produce the smallest reconstruction error. Finally, the quality of the clustering is also demonstrated as the reconstruction errors suggest around 5 feet distance at each frame.

We reach similar conclusions for ball and player trajectories of various time-windows (1-5 seconds) and thus demonstrate that the encoded representation can be learnt from partial data.

[7]Playoff months, May and June, are not included as there are much fewer games in these months.

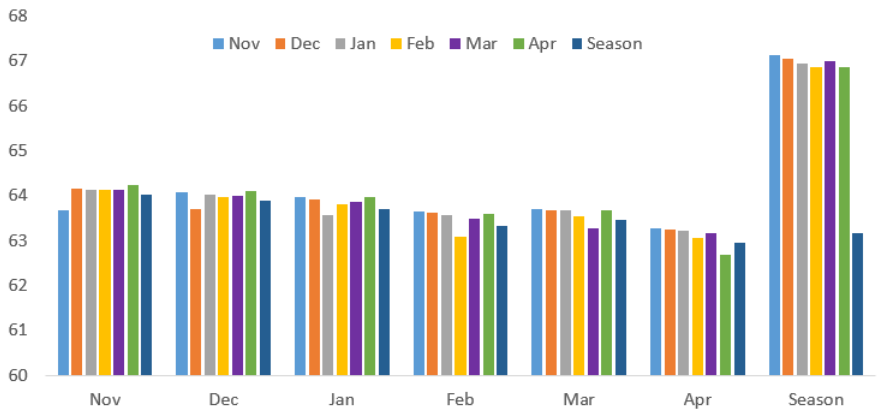


Figure 4.11. Average reconstruction errors for 5-second ball trajectories (in ft)

To further determine the minimum amount of data required to obtain stable clusters, we perform the same k-means clustering on increasing numbers of games and compute the reconstruction errors of the full data. As shown in Figure 4.12, for ball trajectories from 1 to 5 seconds, the average values stabilize once the number of games exceeds 70. Similar patterns are observed on player trajectories.

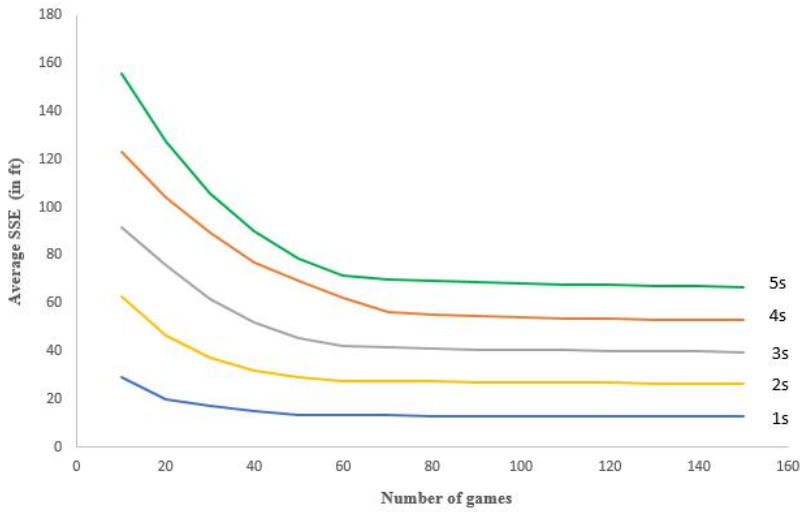


Figure 4.12. Average reconstruction errors against the number of games for ball trajectories

## 4.6. Conclusion

In this chapter, we have developed a search engine for basketball plays that can achieve real-time high quality retrieval from a large number of games. With an advanced similarity metric learnt via a CNN-based autoencoder and a pairwise learning to rank approach, our system is able to adapt to user's specific interest and offer user-specific rankings. To examine its effectiveness, we compared the system's output (predicted) ranking to real user feedback. It would be of interest of follow-up work to extend the system's setting to other sports contexts or even other domains of spatiotemporal tracking data such as animal behavior. The clustering-based encoded representation can be further improved by splitting each cluster with respect to events (e.g., shots, pass, turnover) or by employing a tree-based hierarchical approach.

# References


[1] J. Abara. Applying integer linear programming to the fleet assignment problem. *Interfaces*, 19(4):20–28, 1989.

[2] L. C. Abel. On the ordering of connections for automatic wire routing. *IEEE Transactions on Computers*, 100(11):1227–1233, 1972.

[3] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–26. ACM, 2006.

[4] Eugene Agichtein and Zijian Zheng. Identifying best bet web search results by mining past user behavior. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 902–908. ACM, 2006.

[5] Christoph Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proceedings of the 2000 International Symposium on Physical Design*, pages 19–25. ACM, 2000.

[6] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Action classification in soccer videos with long short-term memory recurrent neural networks. In *International Conference on Artificial Neural Networks*, pages 154–159. Springer, 2010.

[7] C. Barnhart, T. S. Kniker, and M. Lohatepanont. Itinerary-based airline fleet assignment. *Transportation Science*, 36(2):199–217, 2002.

[8] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[9] M. E. Berge and C. A. Hopperstad. Demand driven dispatch: A method for dynamic aircraft capacity assignment, models and algorithms. *Operations Research*, 41(1):153–168, 1993.

[10] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, and Iain Matthews. Win at home and draw away: automatic formation analysis highlighting the differences in home and away team behaviors. In *Proceedings of the 8th Annual MIT Sloan Sports Analytics Conference*. Citeseer, 2014.

[11] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, Sridha Sridharan, and Iain Matthews. Large-scale analysis of soccer matches using spatiotemporal tracking data. In *2014 IEEE International Conference on Data Mining*, pages 725–730. IEEE, 2014.

[12] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. ACM, 2005.

[13] M. Burstein and R. Pelavin. Hierarchical wire routing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2(4):223–234, 1983.

[14] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193. ACM, 2006.

[15] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136. ACM, 2007.

[16] RC Carden, Jianmin Li, and Chung-Kuan Cheng. A global router with a theoretical bound on the optimal solution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(2):208–216, 1996.

[17] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 491–502. ACM, 2005.

[18] Sheng Chen, Zhongyuan Feng, Qingkai Lu, Behrooz Mahasseni, Trevor Fiez, Alan Fern, and Sinisa Todorovic. Play type recognition in real-world football video. In *IEEE Winter Conference on Applications of Computer Vision*, pages 652–659. IEEE, 2014.

[19] Charles Chiang, Majid Sarrafzadeh, and Chak-Kuen Wong. Global routing based on steiner min-max trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(12):1318–1325, 1990.

[20] Charles Chiang, Chak-Kuen Wong, and Majid Sarrafzadeh. A weighted steiner tree-based global router with simultaneous length and density minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(12):1461–1469, 1994.

[21] David Cossock and Tong Zhang. Subset ranking using regression. In *International Conference on Computational Learning Theory*, pages 605–619. Springer, 2006.

[22] Ke Deng, Kexin Xie, Kevin Zheng, and Xiaofang Zhou. Trajectory indexing and retrieval. In *Computing with Spatial Trajectories*, pages 35–60. Springer, 2011.

[23] Antoine Deza, Chris Dickson, Tamas Terlaky, Anthony Vannelli, and Hu Zhang. Global routing in VLSI design algorithms, theory, and computational practice. *JCMCC-Journal of Combinatorial Mathematicsand Combinatorial Computing*, 80:71, 2012.

[24] Y. Fan and C. Liu. Solving stochastic transportation network protection problems using the progressive hedging-based method. *Networks and Spatial Economics*, 10(2):193–208, 2010.

[25] Marshall L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981.

[26] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Symvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM Journal on Computing*, 22(5):1035–1052, 1993.

[27] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[28] Fred Glover. Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[29] Fred Glover. Tabu searchpart II. *ORSA Journal on computing*, 2(1):4–32, 1990.

[30] C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi. The fleet assignment problem: solving a large-scale integer program. *Mathematical Programming*, 70(1-3):211–232, 1995.

[31] Li Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[32] Mark Harmon, Patrick Lucey, and Diego Klabjan. Predicting shot making in basketball using convolutional neural networks learnt from adversarial multiagent trajectories. *arXiv preprint arXiv:1609.04849*, 2016.

[33] Masayuki Hayashi and Shuji Tsukiyama. A hybrid hierarchical approach for multi-layer global routing. In *Proceedings of the 1995 European conference on Design and Test*, page 492. IEEE Computer Society, 1995.

[34] Jörg Heisterman and Thomas Lengauer. The efficient solution of integer programs for hierarchical global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(6):748–753, 1991.

[35] Jiang Hu and Sachin S Sapatnekar. A survey on multi-net global routing for integrated circuits. *INTEGRATION, the VLSI Journal*, 31(1):1–49, 2001.

[36] TC Hu and Man-Tak Shing. *A decomposition algorithm for circuit routing*. Springer, 1985.

[37] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142. ACM, 2002.

[38] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 154–161. ACM, 2005.

[39] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems*, 25(2):7, 2007.

[40] Maurice George Kendall. *Rank correlation methods.* Griffin, 1948.

[41] M. R. Kramer and J. Van Leeuwen. *Wire-routing is NP-complete*. Department of Computer Science, University of Utrecht Utrecht, 1982.

[42] Ulrich Ph Lauther. Top down hierarchical global routing for channelless gate arrays based on linear assignment. In *Proceedings of the IFIP International Conference on VLSI*, pages 141–151, 1987.

[43] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 7(3):1–121, 2014.

[44] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems*, pages 897–904, 2007.

[45] O. Listes and R. Dekker. A scenario aggregation–based approach for determining a robust airline fleet composition for dynamic capacity allocation. *Transportation Science*, 39(3):367–382, 2005.

[46] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[47] Bernard Loeffelholz, Earl Bednar, Kenneth W Bauer, et al. Predicting NBA games using neural networks. *Journal of Quantitative Analysis in Sports*, 5(1):1–15, 2009.

[48] Patrick Lucey, Alina Bialkowski, Peter Carr, Yisong Yue, and Iain Matthews. How to get an open shot: analyzing team movement in basketball using tracking data. In *Proceedings of the 8th Annual MIT Sloan Sports Analytics Conference Conference*, pages 1–10, 2014.

[49] Wing K Luk, Paolo Sipala, Markku Tamminen, Donald Tang, Lin S Woo, and Chak-Kuen Wong. A hierarchical global wiring algorithm for custom chip design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(4):518–533, 1987.

[50] Alan McCabe and Jarrod Trevathan. Artificial intelligence in sports prediction. In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 1194–1197. IEEE, 2008.

[51] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning*, pages 775–782, 2010.

[52] Armand McQueen, Jenna Wiens, and John Guttag. Automatically recognizing on-ball screens. In *2014 MIT Sloan Sports Analytics Conference*, 2014.

[53] B. Pampel and U. Brandes. Orthogonal-ordering constraints are tough. *Journal of Graph Algorithms and Applications*, 17(1):1–10, 2013.

[54] Arvind M Patel, Norman L Soong, and Robert K Korn. Hierarchical VLSI routing–an approximate routing procedure. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(2):121–126, 1985.

[55] Prabhakar Raghavan and Clark D Thompson. Multiterminal global routing: a deterministic approximation scheme. *Algorithmica*, 6(1-6):73–82, 1991.

[56] D. Richards. Complexity of single-layer routing. *IEEE Transactions on Computers*, (3):286–288, 1984.

[57] R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, 16(1):119–147, 1991.

[58] Long Sha, Patrick Lucey, Yisong Yue, Peter Carr, Charlie Rohlf, and Iain Matthews. Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 336–347. ACM, 2016.

[59] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Advances in Neural Information Processing Systems*, pages 937–944, 2002.

[60] H. D. Sherali, K. H. Bae, and M. Haouari. Integrated airline schedule design and fleet assignment: polyhedral analysis and Benders' decomposition approach. *INFORMS Journal on Computing*, 22(4):500–513, 2010.

[61] H. D. Sherali, E. K. Bish, and X. Zhu. Airline fleet assignment concepts, models, and algorithms. *European Journal of Operational Research*, 172(1):1–30, 2006.

[62] H. D. Sherali and B. M. P. Fraticelli. A modification of Benders' decomposition algorithm for discrete subproblems: an approach for stochastic programs with integer recourse. *Journal of Global Optimization*, 22(1-4):319–342, 2002.

[63] H. D. Sherali and X. Zhu. Two-stage fleet assignment model considering stochastic passenger demands. *Operations Research*, 56(2):383–399, 2008.

[64] E Shragowitz and S Keel. A global router based on a multicommodity flow model. *INTEGRATION, the VLSI journal*, 5(1):3–16, 1987.

[65] B. C. Smith. *Robust airline fleet assignment*. PhD thesis, Georgia Institute of Technology, 2004.

[66] B. C. Smith and E. L. Johnson. Robust airline fleet assignment: Imposing station purity using station decomposition. *Transportation Science*, 40(4):497–516, 2006.

[67] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 77–86. ACM, 2008.

[68] Tamás Terlaky, Anthony Vannelli, and Hu Zhang. On routing in VLSI design and communication networks. In *Algorithms and Computation*, pages 1051–1060. Springer, 2005.

[69] Kevin Toohey and Matt Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, 2015.

[70] H. Topaloglu and W. B. Powell. Dynamic programming approximations for stochastic time-staged integer multicommodity flow problems. *INFORMS Journal on Computing*, 18(1):31–42, 2006.

[71] D. Wang, D. Klabjan, and S. Shebalov. Attractiveness-based airline network models with embedded spill and recapture. *Journal of Airline and Airport Management*, 4(1):1–25, 2014.

[72] Kuan-Chieh Wang and Richard Zemel. Classifying NBA offensive plays using neural networks.

[73] J. P. Watson and D. L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370, 2011.

[74] Xinyu Wei, Long Sha, Patrick Lucey, Stuart Morgan, and Sridha Sridharan. Large-scale analysis of formations in soccer. In *Digital Image Computing: Techniques and Applications (DICTA), 2013 International Conference on*, pages 1–8. IEEE, 2013.

[75] Kasun Wickramaratna, Min Chen, Shu-Ching Chen, and Mei-Ling Shyu. Neural network based framework for goal event detection in soccer videos. In *7th IEEE International Symposium on Multimedia*, pages 8–pp. IEEE, 2005.

[76] K Winter and Dieter A Mlynski. Hierarchical loose routing for gate arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):810–819, 1987.

[77] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1192–1199. ACM, 2008.

[78] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398. ACM, 2007.

[79] Zhang Zhang, Kaiqi Huang, and Tieniu Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *18th International Conference on Pattern Recognition*, volume 3, pages 1135–1138. IEEE, 2006.

[80] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29, 2015.

[81] Hai Zhou and DF Wong. Global routing with crosstalk constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(11):1683–1688, 1999.

[82] X. Zhu. *Discrete two-stage stochastic mixed-integer programs with applications to airline fleet assignment and workforce planning problems*. PhD thesis, Virginia Tech, 2006.

# APPENDIX A

## A.1. Complete Formulation for Lagrangian Relaxation

Given the notation and decision variables defined in Sections 2.2 and 2.4.2, the complete formulation on a partitioned network for Lagrangian relaxation algorithm reads as follows.

$$
\begin{aligned}
\min \quad & \alpha \sum_{i}^{n} \sum_{m \in \bar{M}_i} \sum_{k \in N_i \cup N_i'} u_{m,k} + \beta \sum_{i}^{n} \sum_{k \in N_i \cup N_i'} \sum_{m_1 \in \bar{M}_i} \sum_{m_2 \in \bar{M}_i} v_{k,m_1,m_2} \\
& + \gamma \sum_{e \in E} o_e + \theta \sum_{i}^{n} \sum_{s \in S_i} \sum_{k \in N_i \cup N_i'} x_{s,k} d(s,k) \\
& + \mu \sum_{i}^{n} \sum_{s_1 \in S_i} \sum_{s_2 \in S_i} \sum_{k_1 \in N_i \cup N_i'} \sum_{k_2 \in N_i \cup N_i'} 1_{\{x_{s_1,k_1} = x_{s_2,k_2} = 1\}} V(s_1,k_1,s_2,k_2) \\
& + \mu \sum_{i}^{n} \sum_{s_1 \in S_i} \sum_{s_2 \in S \setminus S_i} \sum_{k_1 \in N_i \cup N_i'} x_{s_1,k_1} V'(s_1,k_1,s_2) + \nu \sum_{m \in M} \sum_{a \in A} y_{m,a} \text{length}(a)
\end{aligned}
\tag{A.1}
$$

For each $i = 1, 2, ..., n$, we have the following region-wise constraints.

$$
\sum_{k \in N_i \cup N_i'} x_{s,k} = 1 \qquad s \in S_i \tag{A.2}
$$

$$
\sum_{s \in S_i} x_{s,k} \leq 1 \qquad k \in N_i \cup N_i' \tag{A.3}
$$

$$
y_{m,a_1} + y_{m,a_2} \leq 1 \qquad e = \{a_1, a_2\} \in E_i,\ m \in \bar{M}_i \tag{A.4}
$$

(A.5) $$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} + x_{t,k} \qquad k \in N_i \cup N_i',\ m = (s,t) \in M_i$$

(A.6) $$\sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} \leq 1 \qquad k \in N_i \cup N_i',\ m \in M_i$$

(A.7) $$\sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} \leq 1 \qquad k \in N_i \cup N_i',\ m \in M_i$$

(A.8) $$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j}$$

(A.9) $$x_{t,k} + \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} = \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i}$$

(A.10) $$\sum_{m \in \bar{M}_i} y_{m,a_1} + \sum_{m \in \bar{M}_i} y_{m,a_2} - 1 \leq Lo_e \qquad e = \{a_1, a_2\} \in E_i$$

(A.11) $$y_{m,r} + y_{m,h} - 1 \leq u_{m,k} \qquad k \in N_i \cup N_i',\ m \in \bar{M}_i,\ r \in I_1^i(k), h \in O_2^i(k) \text{ or } r \in I_2^i(k), h \in O_1^i(k)$$

(A.12) $$y_{m_1,r_1} + y_{m_1,h_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{k,m_1,m_2} \qquad k \in N_i,\ m_1, m_2 \in \bar{M}_i,\ r_1 \in I_1^i(k), h_1 \in O_1^i(k)$$

$$r_2 \in I_2^i(k), h_2 \in O_2^i(k) \text{ or}$$

$$r_1 \in I_2^i(k), h_1 \in O_2^i(k), r_2 \in I_1^i(k), h_2 \in O_1^i(k)$$

Let $c = (p,q) \in A_{i,j}$ be an inter-region arc going from region $i$ to $j$ and $c' = (q,p) \in A_{j,i}$ be the corresponding inter-region arc going from region $j$ to $i$, $p \in N_i'$, $q \in N_j'$.

(A.13) $$x_{s,p} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} = y_{m,c} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j}$$

(A.14) $$x_{t,p} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} = y_{m,c'} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i}$$

(A.15) $$y_{m,c} + \sum_{a \in O_1^i(p) \cup O_2^i(p)} y_{m,a} \leq 1 \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j}$$

(A.16) $$y_{m,c'} + \sum_{a \in I_1^i(p) \cup I_2^i(p)} y_{m,a} \leq 1 \qquad m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i}$$

(A.17) $$y_{m,c} + y_{m,r} - 1 \leq u_{m,p} \qquad m \in \bigcup_{j \in adj(i)} M_{i,j},\ r \in I_1^i(p) \text{ or } I_2^i(p)$$

(A.18) $$y_{m,c'} + y_{m,r} - 1 \leq u_{m,p} \qquad m \in \bigcup_{j \in adj(i)} M_{j,i},\ r \in O_1^i(p) \text{ or } O_2^i(p)$$

(A.19) $$y_{m_1,c} + y_{m_1,r_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{p,m_1,m_2} \qquad m_1 \in \bigcup_{j \in adj(i)} M_{i,j},\ m_2 \in \bar{M}_i,$$

$$r_1 \in I_1^i(p),\ r_2 \in I_2^i(p),\ h_2 \in O_2^i(p) \text{ or}$$

$$r_1 \in I_2^i(p),\ r_2 \in I_1^i(p),\ h_2 \in O_1^i(p)$$

(A.20)

$$y_{m_1,c'} + y_{m_1,r_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{p,m_1,m_2} \qquad m_1 \in \bigcup_{j \in adj(i)} M_{j,i},\ m_2 \in \bar{M}_i,$$

$$r_1 \in O_1^i(p),\ r_2 \in I_2^i(p),\ h_2 \in O_2^i(p) \text{ or}$$

$$r_1 \in O_2^i(p),\ r_2 \in I_1^i(p),\ h_2 \in O_1^i(p)$$

The following global constraints are imposed on all the regions.

(A.21) $$\sum_{m \in M} y_{m,a_1} + \sum_{m \in M} y_{m,a_2} - 1 \leq Lo_e \qquad e = \{a_1, a_2\} \in E \backslash \bigcup_i E_i$$

(A.22) $$x_{s,k},\ y_{m,a},\ u_{m,k},\ v_{k,m_1,m_2},\ o_e\ \in \{0,1\}$$

## A.2. Complete Formulation for Progressive Hedging

Similar to Appendix A.1, given the notation and decision variables defined in Sections 2.2 and 2.4.3, the complete formulation for the progressive hedging (PH) algorithm is listed below. Unlike the Lagrangian relaxation approach in which inter-region arcs do not belong to any region, the PH approach partitions the network in such a way that each region includes inter-region arcs either originating from or going into it.

$$\min \quad \sum_{i}^{n} Q_1^i + 0.5 \sum_{i}^{n} Q_2^i \tag{A.23}$$

where

$$\begin{aligned} Q_1^i = {} & \alpha \sum_{m \in \bar{M}_i} \sum_{k \in N_i} u_{m,k} + \beta \sum_{k \in N_i} \sum_{m_1 \in \bar{M}_i} \sum_{m_2 \in \bar{M}_i} v_{k,m_1,m_2} \\ & + \gamma \sum_{e \in E_i} o_e + \theta \sum_{s \in S_i} \sum_{k \in N_i} x_{s,k} d(s,k) \\ & + \mu \sum_{s_1 \in S_i} \sum_{s_2 \in S_i} \sum_{k_1 \in N_i} \sum_{k_2 \in N_i} 1_{\{x_{s_1,k_1} = x_{s_2,k_2} = 1\}} V(s_1,k_1,s_2,k_2) \\ & + \mu \sum_{s_1 \in S_i} \sum_{s_2 \in S \setminus S_i} \sum_{k_1 \in N_i} x_{s_1,k_1} V'(s_1,k_1,s_2) \\ & + \nu \sum_{m \in \bar{M}_i} \sum_{a \in A_i} y_{m,a} \text{length(a)}, \\ Q_2^i = {} & \gamma \sum_{e \in \bar{E}_i \setminus E_i} o_e + \nu \sum_{m \in \bar{M}_i \setminus M_i} \sum_{a \in \bar{A}_i \setminus A_i} y_{m,a} \text{length(a)}. \end{aligned}$$

For each $i = 1, 2, ..., n$, we have the following region-wise constraints.

$$\sum_{k \in N_i} x_{s,k} = 1 \qquad s \in S_i \tag{A.24}$$

(A.25) $$\sum_{s \in S_i} x_{s,k} \leq 1 \qquad k \in N_i$$

(A.26) $$y_{m,a_1} + y_{m,a_2} \leq 1 \qquad e = \{a_1, a_2\} \in E_i,\ m \in \bar{M}_i$$

(A.27) $$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} + x_{t,k} \quad k \in N_i, m = (s,t) \in M_i$$

(A.28) $$x_{s,k} + \sum_{a \in I_1^i(k) \cup I_2^i(k)} y_{m,a} = \sum_{a \in O_1(k) \cup O_2(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{i,j}$$

(A.29) $$x_{t,k} + \sum_{a \in O_1^i(k) \cup O_2^i(k)} y_{m,a} = \sum_{a \in I_1(k) \cup I_2(k)} y_{m,a} \qquad k \in N_i,\ m = (s,t) \in \bigcup_{j \in adj(i)} M_{j,i}$$

(A.30) $$\sum_{a \in O_1(k) \cup O_2(k)} y_{m,a} \leq 1 \qquad k \in N_i,\ m \in \bar{M}_i$$

(A.31) $$\sum_{a \in I_1(k) \cup I_2(k)} y_{m,a} \leq 1 \qquad k \in N_i,\ m \in \bar{M}_i$$

(A.32) $$\sum_{m \in \bar{M}_i} y_{m,a_1} + \sum_{m \in \bar{M}_i} y_{m,a_2} - 1 \leq Lo_e \qquad e = \{a_1, a_2\} \in \bar{E}_i$$

(A.33) $$y_{m,r} + y_{m,h} - 1 \leq u_{m,k} \qquad k \in N_i,\ m \in \bar{M}_i,\ r \in I_1(k),\ h \in O_2(k) \text{ or } r \in I_2(k),\ h \in O_1(k)$$

(A.34)

$$y_{m_1,r_1} + y_{m_1,h_1} + y_{m_2,r_2} + y_{m_2,h_2} - 3 \leq v_{k,m_1,m_2} \qquad k \in N_i,\ m_1, m_2 \in \bar{M}_i,\ r_1 \in I_1(k),\ h_1 \in O_1(k)$$

$$r_2 \in I_2(k),\ h_2 \in O_2(k) \text{ or}$$

$$r_1 \in I_2(k),\ h_1 \in O_2(k),\ r_2 \in I_1(k),\ h_2 \in O_1(k)$$

The following global constraints are imposed on all the regions. For each $i = 1, 2, ..., n$, we need to impose the following conditions to make the scenario-dependent solutions implementable. We have

(A.35)
$$y^i_{m,a} = y^j_{m,a} \qquad j \in adj(i),\ a \in A_{i,j},\ m \in \bigcup_j M_{i,j},$$

where $y^k_{m,a}$ is the value corresponding to $y_{m,a}$ from solving the subproblem in region $k$, $k = 1, 2$.

(A.36)
$$x_{s,k},\ y_{m,a},\ u_{m,k},\ v_{k,m_1,m_2},\ o_a \ \in \{0, 1\}$$