2022

Building Competent Teams of Experts Based on Project Completion Time and Skill Levels

Yalda Yazdanpanah
University of Windsor

Building Competent Teams of Experts Based on Project Completion Time and Skill Level

By

Yalda Yazdanpanah

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2022

Building Competent Teams of Experts Based on Project Completion Time and Skill Level

by

Yalda Yazdanpanah

APPROVED BY:

October 3, 2022

DECLARATION OF ORIGINALITY

ABSTRACT

With many companies quickly expanding their sizes, building the best team of experts from the applicants has evolved into an interesting subject for computer-aided decision-making tasks. In this regard, the Team Formation Problem (TFP) has been well-studied in Artificial Intelligence and operations research literature in recent years. We consider a Team Formation Problem of assigning qualified experts to a given set of positions in a given set of projects where each position is to be filled with an expert with a required skill. In our setting, an expert can be quantitatively characterized by one level per skill, and each expert has a limited workload capacity at any moment of the time. Under the condition that all projects need to be completed before they are due, the ultimate goal is to maximize the gain from all the projects in terms of the overall skill levels of each position. We formulated the problem in Integer Linear Programming (ILP) model. We also designed and implemented two improvement-based heuristic approaches, both following the local search strategy. The first one explores the neighbourhood of the current solution considering both the feasible and the infeasible solutions, where the evaluation of the solutions is defined by a linear combination of the objective function and the number of violated constraints. The second one explores the neighbourhood for only feasible solutions. The solutions obtained from these heuristic approaches and the one obtained from ILP solver Gurobi are compared according to their execution times and objective values.

Keywords: Integer linear programming, Team formation problem, Heuristic, Local Search

DEDICATION

For children of war and children who suffer poverty.

You deserve to be careless, roam in your limitless imagination, and mess around with silly science experiments. You deserve to be children!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

TFP	Team Formation Problem
TFP-SN	Team Formation Problem in Social Networks
LP	Linear Programming
MIP	Mixed Integer Programming
ILP	Integer Linear Programming
KP	Knapsack Problem
KCI	Kolbe Conative Indices
SA	Simulated Annealing
B&B	Branch and Bound
BFS	Breadth First Search
DFS	Depth First Search
FIFO	First In First Out
LIFO	Last In First Out
LS	Local Search
LNS	Large Neighbourhood Search

CHAPTER 1

Introduction

1.1 Overview

Many companies, including large corporations and start-ups, are expanding at a fast pace. Hiring the most competent group of experts has become more difficult for employers due to new objectives, such as the extensive pool of applicants compared to the past. Over the years, the structure of private organizations has changed worldwide, transitioning to a multi-task team-based fashion [9]. The Team Formation problem (TFP) has become a well-known problem in recent years in Artificial Intelligence and operations research literature. "This problem is relatively recent compared to other more classic OR problems" [13]. The objective is to answer the following questions: Who should be hired? How can a decision be evaluated? What influences the gain of a project?

A team is formed by a number of people who come from different fields, master complementary skills, and work together to complete a common task [14]. Although there are some arguments about teams' efficiency [3], there is plenty of evidence that high-performance teams add value to their organizations; especially when decision-making and complex problem-solving are needed.

Generally, the recruitment process includes finding the right team of experts possessing specific skills to maximize the gain of a set of projects. The definition of the *right team* and *project's gain* vary among studies. For example, the *right team* can refer to the most cost-efficient or experienced teams in different contexts. Similarly, maximizing the project's gain can mean maximizing one or several positive social attributes. In particular, we consider a recruitment company (i.e. LinkedIn) that has to optimally assign competent teams to match a required set of skills in several projects while guaranteeing the completion of all projects in their specified timelines. We intend to develop the reputation of all project outcomes by maximizing the skilled experts' recruitment in this research.

1.2 Motivation

The work dynamics change throughout time. Nevertheless, significant changes happened during the COVID-19 pandemic since many businesses were forced to shut down for more than two years in many countries. Some pre-existing limitations have lifted with companies switching to working remotely from home. Since the location was not an issue anymore, companies started to consider a wider pool of candidates for a job. This indicates the demand for a merit-based optimization that finds multiple teams of experts among a large pool of candidates. It also emphasizes the importance of our research.

Many studies on the TFP subproblems focus on maximizing profit by minimizing the cost. In most cases, the employee's salary is a part of the overall cost. Offering a job with the bottom range of salary to an expert can lead to employee dissatisfaction or low work quality, which deflects the primary goal. To overcome this problem, in this thesis, we are proposing an approach that does not focus on the hiring salary. We want to consider each expert's merits and design an optimization solution that assigns the most skilled teams of experts to the available positions on different projects. Companies can assess their overall costs using available databases and machine learning tools to predict an acceptable salary range based on the country's regulations and the job position. For example, Glassdoor [1] is a website where current and former employees anonymously review companies and submit their actual salaries. They release a Job Market Report monthly as a part of their data collection and analysis.

[1]Glassdoor. 2022. Glassdoor Economic Research. [online] Available at: ¡https://www.glassdoor.com/research/¿ [Accessed 1 August 2022].

Companies should be encouraged to schedule their budget and project specifications more realistically before starting the recruitment process.

To the best of our knowledge, few studies in the TFP area take project completion time into account [5]. To have a realistic setup, we think addressing time in team formation problems is crucial although many well-developed studies have ignored it. With that in mind, as shown in Figure 1.3.1, our problem setting can be illustrated by two bipartite graphs, one between the experts and the projects; and the other between projects and the time intervals.

1.3 Contributions

In our setting, we are given a set of projects where each requires a specific set of skills and must be completed by a particular time. Moreover, we are given a set of candidates where each individual has an acquired skill set and a quality score defining each skill level. Each expert has a limited workload, introducing constraints on the number of tasks a person can take simultaneously. This leads to more realistic team construction, higher employee satisfaction, and a better quality of product.

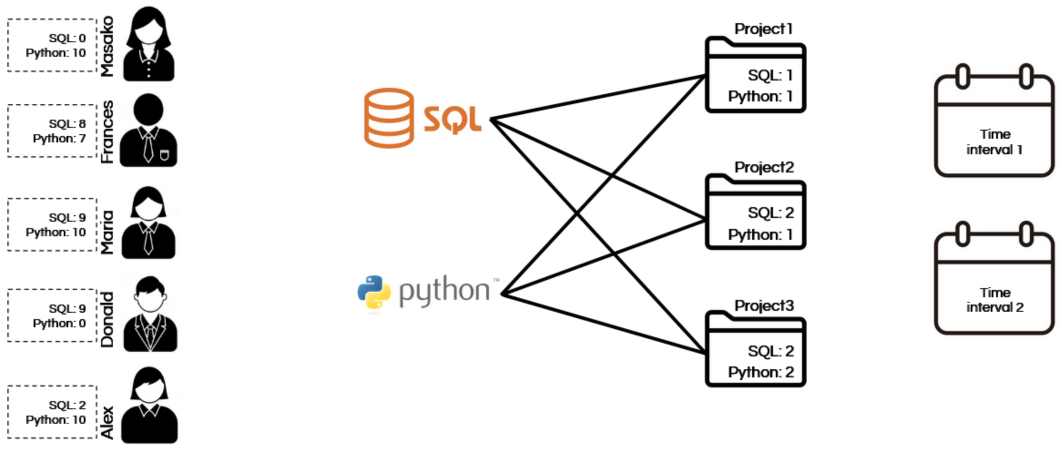


Fig. 1.3.1: An example of our setting with five experts in the pool, three projects, and two time-intervals

For example, Figure 1.3.1 shows a small sample of our problem. We have three projects to complete, each specifying how many experts they need for the two skills, SQL and Python. Note that projects are not mandated to have all the skills. In this example, the projects share the same skills required for their completion. The optimization goal is to hire the most highly skilled experts among the applicants to maximize the gain of each project. The decision can get more complicated due to a vast pool of candidates and projects with overlapping development times. In this example, the number of projects is greater than the time intervals, pointing out that at least two projects will be in conflict. As a result, it adds more constraints to our problem since it exceeds the expert's workload if one takes two jobs scheduled simultaneously.

In summary, we have the following assumptions in our setup:

- There is a set of projects and our goal is to find optimal recruitment decisions for all of them simultaneously.
- Each project has a specification denoting the project completion time, the skills required to complete the project, and the number of experts needed for each skill.
- Each expert's workload capacity is limited to one. An expert can be assigned to at most one skill in each project. They can be assigned to skills in multiple projects as long as those projects are not scheduled on the same timeline.
- Each expert possesses a set of skills, along with a quality score measured for each skill.
- The quality score is a metric to evaluate an expert's success; it is derived from the expert's education, previous work experience, and the number of reviews.

Primarily, we model the problem and utilize Integer Linear Programming (ILP) to formulate an optimal solution for the specified problem. Furthermore, we seek to compare the solution from Gurobi to some alternative heuristic approaches.

1.4 Structure

Further in this thesis, in Chapter 2, several fundamental concepts and ideas related to existing techniques and methodologies are presented in more detail, along with details about various tools and methods to tackle the problem. In Chapter 3, we present the problem of building teams of experts based on skill level and project deadlines, along with our proposed ILP and heuristic solutions. Chapter 4 demonstrates some experiments with synthetic and real-life data and compares the two approaches. Finally, chapter 5 contains some final comments and potential future steps related to this work.

CHAPTER 2

Related Work

Our research belongs to one of the team formation problems. In this chapter, we go over several fundamental concepts and techniques that can be utilized to develop its optimal and close to optimal solutions. TFP is a broad subject; in this chapter, we revisit the literature for those previously-studied TFP problems.

2.1 Fundamental Concepts and Techniques

It is proven by Lappas et al. in [17] that the single team formation is classified as NP-hard. Similar to many other computationally complex problems, there are two effective ways to tackle optimization problems:

- Finding an optimal solution using **Exact Algorithms**.
- Finding a close to the optimal solution using **Heuristic Approaches**.

2.1.1 Integer Linear Programming (ILP)

Linear programming (LP) is a method to achieve an optimal solution for a linear objective function, subject to linear equality and inequality constraints. Linear programming is a particular case of **mathematical programming**. If some or all decision variables are discrete, the problem is known as **Mixed-Integer Programming (MIP)** and **Integer Linear Programming (ILP or IP)**, respectively.

Similar to linear programming, the constraints must be in linear form. Assuming

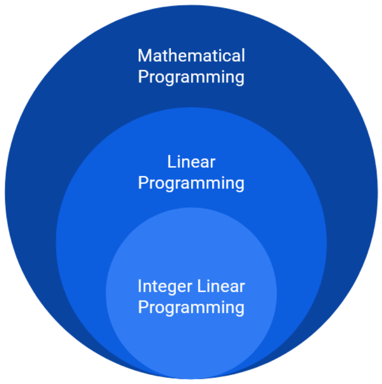


Fig. 2.1.1: Mathematical Programming Hierarchy

x_i is a binary decision variable, some common constraint types are [25]:

$$\text{Set Partition: } \sum x_i = 1; \quad x_i \in \{0,1\}$$
$$\text{Set Covering: } \sum x_i \geq 1; \quad x_i \in \{0,1\}$$
$$\text{Set Packing: } \sum x_i \leq 1; \quad x_i \in \{0,1\}$$

Examples of ILP

In the following, we take a glimpse at two famous problems that share some resemblance with the problem studied in this thesis. The first one is a maximization, modelled at its most basic form. The second is a minimization model we built under assumptions to experiment with conflict handling; such problems can vary depending on the optimization goals.

Knapsack Problem (KP)

The knapsack problem, which first appeared in [20], is to make an optimal decision about packing a set of items, with given values and sizes (i.e. weights or volumes), into a container with a maximum capacity. KP can be modelled into a **0-1 (Binary)**

IP form as follow:

$$\begin{aligned} &\max && \sum_{i=1}^{n} v_i x_i && && (1)\\ &\text{subject to} && \sum_{i=1}^{n} w_i xi \leq W && x_i \in \{0,1\} && (2) \end{aligned}$$

Where the parameters are: W, denoting the knapsack's capacity; v_i and w_i, denoting the value and weight of item i, respectively. The binary decision variable x_i represents the selection of item i for the knapsack; thus, the objective function is to maximize the sum of the chosen items' values. This example is relatively small; the only constraint introduced in (2) limits the decision variables such that the summation of the selected items' weights is always less than or equal to the knapsack's capacity.

Elementary Shortest Path Problem (ESPP)

In a directed graph $G = (V, E)$ with arbitrary costs on the edges in E, the elementary shortest path problem is to find a path between two nodes s and t with the minimum cost. A path is *elementary* if it does not visit any node in V mode than once. A standard integer programming formulation [27] can be found in the following:

$$\begin{aligned} &\min && \sum_{(i,j)\in E} c_{ij} x_{ij} && (1)\\ &\text{subject to} && \sum_{(i,j)\in\delta^+(i)} x_{ij} - \sum_{(j,i)\in\delta^-(i)} x_{ji} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{else} \end{cases} \qquad \forall i \in V && (2)\\ & && \sum_{(i,j)\in\delta^+(i)} x_{ij} \leq 1 \qquad \forall i \in V && (3)\\ & && x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E && (4) \end{aligned}$$

Where the parameter is: $c_{ij} \in \mathbb{R}$ denoting the edge costs (weights). The binary decision variable x_{ij} represents the selection of edge (i, j) for the path. $\delta^+(i)$ and $\delta^-(i)$ denote the set of outgoing and incoming edges of node i. Constraints (2) are related to flow conservation; the equality depends on the node i (i.e. the left-hand side is equal to 1 if node i is the starting point). Constraints (3) guarantee that each edge's outgoing degree is at most one.

2.1.2 Branch and Bound Algorithm (B&B)

Branch and bound is an exact algorithm [16] for global optimization of various problems including ILP models[18]. B&B represents the solution as a **state space tree**; it evaluates all possible permutations by storing partial solutions called subproblems in a tree structure.

The fundamental components of the B&B are [21]:

- **The search strategy:** The order in which subproblems in the tree are explored, which is usually **breadth-first search (BFS)**. To name a few other options, we can mention depth-first search (DFS), best-first search, and cyclic best-first search.

- **The branching strategy:** How the solution space is partitioned to produce new subproblems in the tree. For instance, depending on the container used for the *next node* exploration, B&B can be classified into **FIFO-BB**, **LIFO-BB**, **(LC-BB)** [1].

- **The pruning rules:** Rules that prevent exploration of suboptimal regions of the tree.

Deterministic algorithms seem promising in finding the optimal solution. Nonetheless, their execution time might get very large in many cases since they consider every

[1]The containers used for the First-In-First-Out (FIFO) B&B and the Last-In-First-Out (LIFO) B&B are queue and stack, respectively. Lastly, the Least-Cost-First search B&B expand paths from the frontier in order of their costs.

combination, which can be their major downside.

2.1.3 Greedy Algorithms

Using *dynamic programming* to determine the best decisions can be an overkill for many optimization problems. Therefore, **greedy algorithms** are developed to make a sequence of decisions, each one being in some way the best fit at the moment. By choosing the best decision, not necessarily globally optimum, at each step, they produce locally optimal solutions that approximate a globally optimal solution in an appropriate amount of time [7].

The solution constructed by the end of a greedy approach is the final solution; no improvements are involved. They are versatile, straightforward, and efficient. However, they do not necessarily yield a globally optimal solution. Another downside of greedy algorithms is that they can not solve optimization problems with negative graph edges. In the latter sections of the current chapter, we will go through some applications of the greedy approach over previous studies related to our work. Before moving forward, let us summarize the steps involved in a greedy algorithm:

1. Determine the optimal structure of the problem and find the best subproblem.
2. Determine the objective function, which evaluates the best choice at each step.
3. Develop an iterative or recursive process to inspect all subproblems and create an optimum solution.

2.1.4 Meta-Heuristic Algorithms

Meta-Heuristic algorithms are computational intelligence paradigms primarily used for complex optimization problems [1]. Similar to the greedy approach, the original intention of a heuristic algorithm is to reduce the search space and lower the run-time at the expense of some potential optimality loss. Heuristic algorithms use probability and statistics to avoid checking all the combinations in the solution region. Note that, unlike greedy algorithms, they do not construct a solution; they usually start from a random solution and then enhance it with each step until reaching the local/global optimum. These iterations carry until the system converges and the solution found meets some predefined criterion [31].

Meta-heuristic algorithms imitate natural evolutionary principles to perform the search and optimization procedures. These methods choose their path through the parameter space using random factors [24]. As the name suggests, these methods are a higher-level heuristic that is more general in problem-solving, making these techniques powerful.

Simulated Annealing

Inspired by the annealing procedure, a metal's heating and slow cooling, simulated annealing was proposed by [15] for the first time. Generally, in the beginning, the algorithm explores successors wildly randomly. This pace goes down as time passes until there is a time when things are settled.

At each annealing temperature, the SA algorithm generates a new potential solution (or neighbour of the current state) to the problem considered by altering the current state according to a predefined criterion. Then, the new state's acceptance is based on the satisfaction of the Metropolis criterion, and this procedure is iterated until convergence. The basic idea behind the Metropolis criterion is to avoid getting trapped in extreme local optima. SA is problem-independent.

The main difference between SA and other heuristic approaches like **stochastic hill climbing** remains in finding the neighbourhood transition methods. In stochastic hill climbing, the steps are taken randomly, and the current point is replaced with a new point, provided the new point is an improvement to the previous point. The next node is constructed in simulated annealing by altering the current node while the search works the same way. However, the worst points are sometimes accepted to allow the algorithm to learn answers that are eventually better. Furthermore, SA is often used when the search space is discrete, and it seems a good fit for our problem - which will be described more in detail in Chapter 3.

2.2 Team Formation Problem (TFP)

As mentioned in the introduction, the Team formation problem generally refers to finding team members from a pool of candidates that would form a team of maximum "efficacy" to take on a task. The two major fields of interest in this problem are: **Operations Research (OR)** and **Data Mining (DM)**. In the OR version, the drive is the organization's demands [32], and the goal is to identify teams of candidates whose skills match the job positions the best. In the DM version, the drive is the social networks [17], and the goal is to identify closely acquainted candidates whose combined skill-set equals the task's demanded skills.

Principally, the previous studies on the TFP can be classified as assignment-based and community-based TFPs [13]. The **assignment-based** category involves a bipartite matching between the set of candidates and the set of jobs (or tasks), each carrying a suitability score. The mentioned weight or score suggests the candidate's compatibility with a job position. Then, the objective is to maximize the suitability score matching. Examples of this can be found in [4], where Anagnostopoulos et al. investigate their proposed TFP model using various scoring functions such as *all skills required*, *least-skill dominant*, *fraction of skills possessed*, *micro average of skill*, and *macro average of skill.* Another example is using Kolbe Conative Indices (KCI) as a parameter in the scoring measure in [9].

On the other hand, the **community-based** category concentrates more on the previous relationships among the experts [30, 6], for example, acquaintance, collaborations, and closeness. Therefore, it involves a minimum spanning tree (MST) or a sub-graph over the input graph describing the communication cost. Then, the objective is to minimize such costs or maximize the compatibility if the weights are defined positively.

Majumder et al. pioneer exploring the TFP in a software development context using the GitHub collaboration graph as a dataset for experimentation [19]. They have crawled millions of software repositories spanning four years and hundreds of thousands of developers from GitHub [2]. This data collection can be part of a somewhat reliable scoring measure, though it is primarily beneficial for building the communication graph in their work. Thus, a community-based formulation with binary skills is proposed, along with the difference between a modification on the original covering constraint and an additional packing constraint.

In comprehensive research regarding TFPs [13], Juárez et al. point out that the communication graph in many community-based models is considered an undirected weighted graph, which shows the one-on-one relationship between two candidates. The pairwise interactions might differ from the dynamics of a group of three, which much of the literature disregards.

Although the community-based models might seem more realistic, we must take a step back and review the assumptions. All TFPs assume the candidates as an idling resource [13], which is usually not the case in real life. Our work addresses an application of TFPs that aims to form groups of freelancer or contract-based employees for projects. An actual scenario would suggest that a company would rule out the good connections from the optimization problem if existent. Then the remaining decision-making for the company will be cast into an assignment form.

Regardless of the mentioned categories, the problem setting can involve forming a single or many teams. The goal in **single team formation** is to generate one team; such problems are relatively basic versions of **many teams formation**. In a study on balanced teams [29], Van de Water et al. analyzed dividing a pool of students into teams where each team has nine roles, and each student has a suitability score for each role. The objective of maximizing the suitability score was subjected to constraints based on students' capacities and a distinctive set of balancing constraints that, for example, restricted students with a high score on the same role from being in the same team.

The many team formation can be reduced to a single team formation if candidates are bounded to be part of only a single team. Fitzpatrick et al. proposed a MIP formulation [9], Labour Pool Extraction Model - Multiple Teams (LPEMT), and

[2]a popular open-source social coding network

a heuristic algorithm to form multiple teams from a pool of candidates with the necessary skills. The objective function of their mathematical model computes the deviation from optimal Kolbe Conative Indices across all teams. KCI measures an individual's instinctive behaviour or drive, which does not imply either good or bad; merely a means to classify and understand the reasons we do things the way we do or react to things in certain ways.

Sukthankar et al. formulate a TFP where each agent (i.e. candidates or experts) is allowed a single team assignment. So, agents already in a team are constrained to move according to a set of team behaviours. The offered algorithm is called Simultaneous Team Assignment and Behavior Recognition (STABR) [26] that generates behaviour annotations from spatio-temporal agent traces. Their evaluation metric is based on team assignment accuracy, behavioural recognition accuracy, and hypothesis set size. One of their benchmarks was a military simulation, which is not aligned with our goal but shows how diverse the applications of TFP can be.

CHAPTER 3

Proposed Methods

We have already outlined the details of the TFP problems. This section formally describes our setting and provides a comprehensive view of our proposed methods.

3.1 Preliminaries

3.1.1 Notation and Setting

The notations described in this section are used in all the methods discussed in future sections. The terms mentioned in Chapter 2 are defined as follow:

Skills. We consider a set $\mathcal{S}$ of skills, which can be any qualification a candidate can have, such as *Python*, *SQL* or *UI/UX design*. $|\mathcal{S}|$ denotes the cardinality of S.

Experts. We consider a colossal set $\mathcal{E}$ of experts (or candidates) who are seeking to be hired. Each expert $e \in \mathcal{E}$ possesses a subset of acquired skills S'_e, such that $S'_e \subseteq \mathcal{S}$. For simplicity, this information is embedded in each expert's quality measure: $Q_{es}; e \in \mathcal{E}, s \in \mathcal{S}$. So instead of S'_e, we keep $Q_{es} \in \{0, 1, ..., n\}$ for each pair of (e, s). $Q_{es} = 0$ denotes that expert e does not have skill s at all; if this number equals to any integer between 1 to n, it denotes expert e's possession and level of proficiency for skill s. Each expert has a workload capacity limited to one, showing that each one can be assigned to at most one task [1] in a particular time interval.

Projects. We consider a set of $\mathcal{P}$ of projects, $PSR_{sp}, s \in \mathcal{S}, p \in \mathcal{P}$ is an integer $\in \{0, 1, 2, ...\}$ assessing the number of experts project p needs for skill S. Like the experts, we embedded the subset of skills each project requires in PSR, so for instance,

[1] A *task* refers to a particular skill in a project.

$PSR_{sp} = 4$ means project p needs four experts possessing skill s in order to complete. If this number equals zero, project p does not need skill s for completion.

Time intervals. We assume that time is a discrete parameter that can refer to specific weeks or months. We consider a set of time intervals $\mathcal{T}$ that projects can be assigned to.

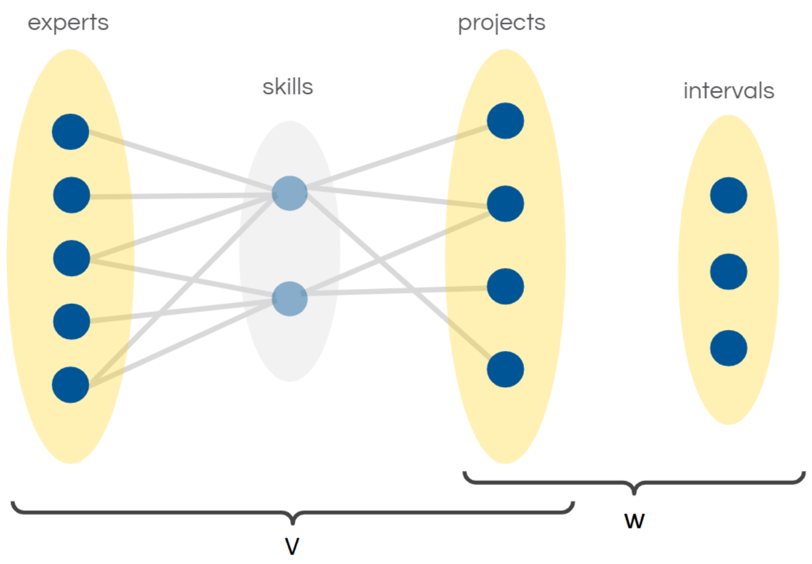


Fig. 3.1.1: The abstract illustration of our studied problem

We can break down the problem into two main subproblems:

Team Assignments: As shown in Figure 3.1.1, for the subproblem of assigning experts to projects, we consider the assignment problem on a bipartite graph G, where nodes N_e and N_p represent expert and project nodes, respectively. Edges E represent possible assignments of experts to specific skills in projects. The grey edges in the illustration show the expert skill quality (Q) and project skill requirements (PSR) associations. These denote what links are available/allowed between the left side of the graph and the right.

Projects Scheduling: For the subproblem of assigning projects to time intervals, we consider the scheduling problem on a bipartite graph G' where N_p and N_t represent project and time interval nodes, respectively. Edges E' represent possible assignments of projects to time intervals.

3.1.2 Problem Description

Problem 1 Consider a set of projects $\mathcal{P}$, each specified with a completion time and a set PSR_{sp}. Given a project $p \in \mathcal{P}$, PSR_{sp} determines the number of experts that project p requires in skill s, where $s \in \mathcal{S}$. We have a pool of experts $\mathcal{E}$. Each given expert $e \in \mathcal{E}$ has a limited workload and is characterized by a quality measure Q_{es} defining the expert's capabilities in skill s. Given all that, the goal is to find a solution so that all the hiring requirements of the projects are satisfied, the total quality score over all the projects is maximized, and the experts are not assigned to overlapping time intervals.

Notation	Description
$\mathcal{E}$	set of experts
$\mathcal{S}$	set of skills
$\mathcal{P}$	set of projects
$\mathcal{T}$	set of time intervals
e	index for experts; $e \in E$
s	index for skills; $s \in S$
p	index for projects; $p \in P$
t	index for time intervals; $t \in T$
PSR_{sp}	number of experts required for skill s in project p
Q_{es}	extracted quality measure of expert e in skill s
W_{pt}	1 if project p is selected for time interval t; 0 otherwise
V_{esp}	1 if expert e is assigned to skill s in project p; 0 otherwise
Z_{pqt}	1 if project p and q are assigned to the same timeslot t; 0 otherwise

Table 3.1.1: Summary of Notations

3.2 ILP Model For TFP with Maximized Quality

In this section, we outline our proposed ILP model and further elaborate over the objective function and constraints.

Objective function:

$$\max \quad \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{e \in \mathcal{E}} (Q_{es} \times V_{esp}) \tag{1}$$

Subjected to:

$$V_{esp} + V_{euq} + \sum_{t \in \mathcal{T}} Z_{pqt} \leq 2 \qquad \forall e \in \mathcal{E}, \forall p, q \in \mathcal{P}, \forall s, u \in \mathcal{S} \tag{2}$$

$$Z_{pqt} \geq W_{pt} + W_{qt} - 1 \tag{3}$$

$$Z_{pqt} \leq W_{pt} \tag{4}$$

$$Z_{pqt} \leq W_{qt} \qquad \forall t \in \mathcal{T}, \forall p, q \in \mathcal{P} \tag{5}$$

$$\sum_{t \in \mathcal{T}} W_{pt} = 1 \qquad \forall p \in \mathcal{P} \tag{6}$$

$$\sum_{e \in \mathcal{E}} V_{esp} \leq PSR_{sp} \qquad \forall s \in \mathcal{S}, p \in \mathcal{P} \tag{7}$$

$$\sum_{s \in \mathcal{S}} V_{esp} \leq 1 \qquad \forall e \in \mathcal{E}, p \in \mathcal{P} \tag{8}$$

As discussed before, the goal is maximize the gain of all projects simultaneously. Our comprehension of gain is the summation of skill quality that the team of experts brings to the project. Thus, as shown in equation (1), the objective function is maximize Q_{es} over all of the selected experts in all projects.

Handling Conflicts

Definition 1 *(Conflict) A conflict Z between two projects $p, q \in \mathcal{P}$ occurs when both of them are assigned to a time interval $t \in \mathcal{T}$. A conflict is represented as a derivative Z_{pqt}.*

Hence, a conflict happens between project p and q when the decision variables responsible for this selection W_{pt} and W_{qt} are assigned to one. In such case, Z_{pqt} is the product of those assignments: $Z_{pqt} = W_{pt} \times W_{qt}$. We can rewrite this constraint to linearize it.

Equations (3, 4, and 5). Now that conflict is formally and mathematically clear; we must manage the consequences of the conflict. If in the given problem we have $|\mathcal{T}| < |\mathcal{P}|$, a *conflict* between two projects is inevitable. Then, we should restrict the experts' assignment so that no expert is selected for more than one skill concurrently in one time interval. As shown in equation (2), for every two skills $u, s \in \mathcal{S}$, and every two projects $p, q \in \mathcal{P}$, expert e can be selected for at most one skill in a project at time interval t. In other words, There is at most one connection (link) between expert e and time interval t.

Project Completion Guarantee

One of the goals of our model is that all projects must be completed. For such a cause, all projects must be assigned to a time interval. So, we have equation (6) that ensures that the summation of all project assignments over time intervals for each project equals 1, which means that there is precisely one time interval to which project p is assigned.

Project's Skill Requirement (PSR)

As shown in Table 3.1.1, PSR_{sp} determines the number of experts required for skill s in project p. In other words, the summation of the selected experts for each skill and each project must reach this number. Note that $\sum_{e \in \mathcal{E}} V_{esp} \geq PSR_{sp}$ would result in hiring abundant experts for the task since the variable V_{esp} is used in the objective function, which is to maximize the summation of the selected experts by their skill level. Thus, assuming the $|\mathcal{E}|$ is large enough, we have $\sum_{e \in \mathcal{E}} V_{esp} \leq PSR_{sp}$ to limit and satisfy the project skill requirements.

Expert's Workload Limitation

By having the constraint (8), we secure that for each expert and project, the summation of all the skills an expert is responsible for is at most 1. Moreover, since a project cannot be assigned to multiple time intervals simultaneously, we guarantee that no expert is assigned to more than one task at a time.

3.3 Steepest Descent Local Search (LS)

Local search is a single-solution-based family of metaheuristics [12]. The emphasis is on exploitation to find a nearly optimal solution within a reasonable computation time. At each iteration of the algorithm, an improving solution is found by searching the *neighbourhood* of the current solution. These neighbourhoods are developed by generating partial yet large enumeration trees based on the current solutions and heuristically searched [2, 23]. The main property that must characterize a neighbourhood is its *locality*. A neighbourhood mapping has a strong locality if neighbours have similar qualities. It helps the local search to perform a meaningful search in the landscape of the problem [28].

The initial solution $\mathcal{A}$ in the search space consists of a fully assigned set of variables of a possible world. Starting from that initial state, we can change any variable to get to a neighbour $\mathcal{A}'$. At each step, only the current solution is kept in the memory. According to the **Steepest Descent** algorithm, we need to evaluate each neighbour using an evaluation function $Eval(\mathcal{A}')$. After exploring the current neighbourhoods, we substitute the current solution $\mathcal{A}$ with the neighbour with the highest evaluation $\mathcal{A}'_{best}$. Below is an abstract overview of steps in the steepest descent local search:

$\boxed{1.}$ Initiate solution $\mathcal{A}$, which is a complete assignment of all decision variables.

$\boxed{2.}$ Move to neighbour $\mathcal{A}' \sim \mathcal{A}$ such that $Eval(\mathcal{A}') < Eval(\mathcal{A})$ [2].

[2]Assuming that the evaluation function $Eval(\mathcal{X})$ refers to the cost of the solution $\mathcal{X}$, the goal is to minimize this cost in a general context.

$\boxed{3.}$ Repeat step 2 until $\mathcal{A}$ is a satisfying solution.

It is common to generate the initial solution randomly. However, discovering a feasible neighbour from an infeasible state can be challenging in some problems like the one studied here. Different techniques can be employed to overcome this problem; we will focus on the following and describe the three main components, **input**, **neighbourhood relations**, and **evaluation function** in each one:

- Searching the feasible region from a **feasible starting solution** with the objective function as the evaluation function.
- Searching feasible and infeasible regions in the search space from an **infeasible starting solution** with a hybrid evaluation function.

3.3.1 LS-Hybrid

We propose *LS-Hybrid*, a heuristic local search with a hybrid evaluation function that traverses feasible and infeasible regions until the local optima is found. After explaining the key components of the algorithm, we present the pseudocode and some discussions.

Input:

As defined in Table 3.1.1, V and W are the decision variables, indicating the team assignments [3] and projects scheduling, respectively. The initial solution $\mathcal{A}$ can be any complete, randomly assigned set of all the variables in V and W. The indexes to the selected time interval and their associated projects are kept in the data structure, *SCHEDULE* for faster access during the local search. For further details, refer to Appendix B.1. This solution may be violating one or more constraints.

Neighbourhood Relations:

Assume that a change in one variable in V or W creates a new neighbour. The variable domains in this discrete optimization problem is $0, 1$ since they are binary.

[3] A *team assignment* refers to the assignment of an expert e to a specific skill s in project p.

Consider the current solution $\mathcal{A}$. We can go to the next neighbour $\mathcal{A}'$ by changing a single variable V_{esp} to its opposite, $1 - V_{esp}$ while all other variables in V and W remain the same as in $\mathcal{A}$. Similarly, by fixing all of the variables to their current values and only changing a single variable W_{pt} to $1 - W_{pt}$ we produce a new neighbour. [4] We will use the term **flipping** for such change.

In this algorithm, the search space includes all possible solutions to the problem. We will traverse parts of the feasible and infeasible solutions in the search space, which can be further decreased by selecting some **hard constraints**. Constraint (6) is the only **linear equality constraint** among all the constraints introduced. We interpret this as a constraint that cannot be compromised to get a feasible solution. Thus, for the W variables, we can prevent multiple assignments of each project p to different time intervals by holding constraint (6) as a hard constraint. It will increase the likelihood of converging into an overall feasible solution. Besides constraint (6), all of the constraints introduced for this problem are **bound constraints**. There is more than one case where inequalities can hold valid. We aim to rigorously explore all permutations of V variables to find any neighbour better than the current one. Therefore, all constraints (2, 3, 4, 5, 7, 8) are considered soft. Note that soft constraints may be violated during the search, which will be assessed with the evaluation function and avoided.

Evaluation Function:

The evaluation function $Eval(\mathcal{X})$ determines the worth of a solution $\mathcal{X}$. In this LS algorithm, we propose a hybrid evaluation function of minimizing constraint violations $Const(\mathcal{X})$ and maximizing the objective function $Obj(\mathcal{X})$, such that the following linear combination holds:

$$Eval(\mathcal{X}) = Obj(\mathcal{X}) - \alpha \times Const(\mathcal{X})$$

[4] The target variable will turn into zero if it already has the value one, and vice versa.

Where α is a weight on the constraint violation, this way, a solution with minimum $Const(\mathcal{X})$ and maximum or same $Obj(\mathcal{X})$ has the priority to get selected as the next neighbour to be traversed. As shown in equation (1), the problem's objective is finding the maximum gain (quality score) over the selected experts, so the $Obj(\mathcal{X})$ calculation is apparent. There are various ways to calculate the $Const(\mathcal{X})$ for constraint violation check, targeting the number of the constraint violations in solution $\mathcal{X}$ or the distance of the solution $\mathcal{X}$ to the optimal solution.

Although our primary goal is to minimize the number of violations, counting the number of constraints violated can be insufficient in many cases. Assume there are two neighbours $\mathcal{X}'$ and $\mathcal{X}''$ that only violate constraint number (7). By proceeding with the violation count, both solutions $\mathcal{X}'$ and $\mathcal{X}''$ will get a fair chance of being the potential best in the neighbourhood. Instead, by calculating the distance, we consider a weight for the violated constraint. For instance, in constraint (7), if the inequality $\sum_{e\in\mathcal{E}} V_{esp} \leq PSR_{sp}$ does not hold for all $s \in \mathcal{S}, p \in \mathcal{P}$, rather than incrementing one, we keep the absolute value of the subtraction $|\sum_{e\in\mathcal{E}} V_{esp} - PSR_{sp}|$ to measure how far the solution is from the nearest feasible one. Generally, in every constraint inequality, the subtraction of the left-hand side by the right-hand side is the distance that a solution $\mathcal{X}$ has until it is within the boundaries.

$$Dist(\mathcal{X}) = |LHS - RHS|$$

Furthermore, since the evaluation needs to be performed when examining each neighbour, it must be computationally inexpensive. It is therefore not viable to re-evaluate all assigned values in V and W of a solution $\mathcal{X}$ for constraint violations. Instead, we can calculate the difference in constraint violation distance between the currently selected solution $\mathcal{X}$ and every neighbour $\mathcal{X}'$. To do that, we identify all the constraints violated by a single expert (in the case of V-flipping) or project (in the case of W-flipping) before the flip action, using a slightly modified function $S\_Const(\mathcal{X}, e, s, p, t)$, where e, s, p, and t are the indexes that flipped. Then, after the flip is performed, we once again calculate the number and amount of constraints violated by the same single expert or project with the new value assignments, $S\_Const(\mathcal{X}', e, s, p, t)$. This way, we can study the effect of the changed indexes on the constraint violations, and the difference $S\_Const(\mathcal{X}', e, s, p, t) - S\_Const(\mathcal{X}, e, s, p, t)$ should be the same as $Const(\mathcal{X}') - Const(\mathcal{X})$.

Discussion:

As shown in LS-Hybrid's pseudocode 3.3.1, the main loop repeats until a maximum number of idle iterations have reached. A satisfactory solution for us is one that is the best among all its neighbours based on the evaluation function. The maximum iteration counter, *iter_counter*, will reset back to its maximum value when a better solution is found; whereas, it decreases each time a neighbour that is not better than the current solution is checked.

One of the advantages of this algorithm is that the neighbour production is speedy and straightforward. The algorithm is not problem-specific and can be adopted non-exclusively by similar problems. The search will not immediately converge to the feasible region, but with the proposed hybrid evaluation function, it finally gets into a locally optimal solution in many cases. The major drawback of the LS with hybrid evaluation function is its large neighbourhood size. The scope of each neighbourhood, which equals to $(|\mathcal{E}| \cdot |\mathcal{S}| \cdot |\mathcal{P}|) + (|\mathcal{P}| \cdot |\mathcal{T}|)$, depends heavily on the input size.

We will experiment LS-Hybrid with a feasible starting point in Chapter 4. The initial random solution can be adjusted to start from a feasible solution (see *Init_G* in Appendix B.2 for more details). Following the same steps as Algorithm 3.3.1, this time, we choose the neighbours with zero violations so that we can stay in the feasible region. Nevertheless, the large number of analyses for each iteration will still grow proportionally with the input size, which is not advantageous.

Algorithm 3.3.1 LS-Hybrid

```
Data: $\mathcal{E}, \mathcal{S}, \mathcal{P}, \mathcal{T}, PSR, Q, max\_iter, \alpha$
Result: $W, V$

$iter\_counter \leftarrow max\_iter$
$V, W, \text{SCHEDULE} \leftarrow Init\_R(\mathcal{E}, \mathcal{P}, \mathcal{S}, \mathcal{T}, PSR)$ // initiate V and W
// get the initial evaluation function:
$obj_{best} \leftarrow Obj(V)$
$const_{best} \leftarrow Const(V, W, p)$
$eval_{best} \leftarrow const_{best} - \alpha \times obj_{best}$
while $iter\_counter > 0$ do
    foreach $p \in \mathcal{P}$ do
        $t_{sel} \leftarrow \text{SCHEDULE}[p]$
        $W_{p,t_{sel}} \leftarrow 0$ // flip the current assignment
        foreach $t \neq t_{sel} \in \mathcal{T}$ do
            $W_{p,t} \leftarrow 1$ // flip another assignment
            $obj_{nbr} \leftarrow Obj(V)$
            $const_{nbr} \leftarrow S\_Const(V, W, e, s, p)$
            $eval_{nbr} \leftarrow const_{nbr} - \alpha \times obj_{nbr}$
            if $eval_{nbr} > eval_{best}$ then
                $W\_flip \leftarrow True$
                update $obj_{best}, eval_{best}$
                best $\leftarrow (p, t_{sel}, t)$
                reset $iter\_counter$
            else
                decrement $iter\_counter$
        $W_{p,t_{sel}} \leftarrow 1$
    foreach $V_{e,s,p} \in V$ do
        $V_{e,s,p} \leftarrow 1 - V_{e,s,p}$ // flipping
        $obj_{nbr} \leftarrow Obj(V)$
        $const_{nbr} \leftarrow S\_Const(V, W, e, s, p)$
        $eval_{nbr} \leftarrow const_{nbr} - \alpha \times obj_{nbr}$
        if $eval_{nbr} > eval_{best}$ then
            $W\_flip \leftarrow False$
            update $obj_{best}, eval_{best}$
            best $\leftarrow (e, s, p)$
            reset $iter\_counter$
        else
            decrement $iter\_counter$
        $V_{e,s,p} \leftarrow 1 - V_{e,s,p}$
    // figuring out which flip led to the selected neighbour,
    // then transitioning to that neighbour
    if $W\_flip = True$ then
        $p, t_{sel}, t \leftarrow$ best
        $W_{p,t_{sel}} \leftarrow 0$
        $W_{p,t} \leftarrow 1$
    else
        $e, s, p \leftarrow$ best
        $V_{e,s,p} \leftarrow 1 - V_{e,s,p}$
```

3.3.2 LNS

We propose a Large Neighbourhood Search (LNS) algorithm with modified neighbourhood structures to overcome the previously mentioned issues with LS-Hybrid. With this method, we aim to limit the search only to the feasible region in the search space. It is challenging to navigate among solutions and avoid getting stuck in a state where all the neighbours are infeasible. Since producing each neighbour via LNS techniques is more computationally expensive than LS-Hybrid, we decided to simplify the evaluation function and consider all of the constraints hard. Similar to the previous section, we explain the key concepts and then provide the pseudocode of our algorithm along with some discussions.

Input

The search space considered for LNS is the feasible region only; thus, all constraints are considered hard constraints in this method. Since we no longer intend to check the constraint violations, the search's starting point must also be in the feasible region. It will not be purely random; however, within the boundaries of the constraints, it is still a random assignment of variables. We temporarily ignore the objective function and employ a greedy method, *Init_G*, to solve a satisfactory version of the problem. We build the feasible solution while generating three data structures, *TEAM*, *SCHEDULE*, *AVAILABLE*, and *IN_COMMON*.

The *SCHEDULE* generated via the greedy initial solution generator differs from the *SCHEDULE* previously used. While scheduling the projects, this time, we keep all projects assigned to a specific time interval in individual lists in *Init_G*. It helps faster and more organized access to the data and guarantees constraint (8) does not get violated.

TEAM, which is also not a part of the solution, helps keep track of task assignments. It holds the list of all experts assigned to each skill in each project (i.e. teams for each skill in all projects). $TEAM_p$ contains all the experts assigned to different skills in project p.

AVAILABLE is a data structure that keeps lists of experts $\subseteq \mathcal{E}$ who are available at each time interval. $AVAILABLE_t$ is the list of experts who are unemployed or assigned to projects in time intervals other than t; thus, we know they are available to be assigned to any task at time t.

IN_COMMON is a graph that indicates if two projects have any experts hired for both. In this graph, where the nodes are all projects, an edge (p, q), where $p, q \in P$, shows that p and q have at least one expert assigned to both projects. If there is no edge between the two projects, they are exclusive regarding expert assignments. Refer to Appendix B.2 for more details about *Init_G* and the rest of the introduced data structures.

Neighbourhood Relation:

As mentioned before, an essential step in the steepest descent local search is finding the best neighbour to the current state that satisfies all the constraints. In the previous approach, we used brute force to find the next neighbour. In contrast, in this algorithm, we utilize constraint programming to explore the neighbourhood since it is relatively large. Because of the characteristics of our problem, we do not want to limit ourselves to small neighbourhoods; instead, we aim to use the large neighbourhood to our advantage and use discrete optimization solving techniques to unravel the overall problem.

Given a solution $\mathcal{A}$, to specify a large neighbourhood, we fix k% of the variables in V and W to their current value. Then, we proceed with solving the remaining unfixed variables. Solving this problem is similar to solving a reduced version of the original problem. To be more precise, we aim to find one pair of interchangeable assignments, fix the rest of the solution, and then swap these two assignments to get a new neighbourhood. The steps required so two experts eA and eB can alternate their tasks, SWAP(V, p, eA, eB, sA, sB), are as follow:

$$V_{eA,sA,p} \leftarrow 0$$
$$V_{eB,sA,p} \leftarrow 1$$

Expert eB can also be chosen from the set of experts available at time t in which p is currently scheduled, $AVAILABLE_t$. In this case, sB would not have any values, and the previous steps are sufficient. Nonetheless, if sB exists, it indicates that the successful swap is selected from within the projects. So, we will have:

$$V_{eB,sB,p} \leftarrow 0$$
$$V_{eA,sB,p} \leftarrow 1$$

Projects are initially assigned to time intervals using the greedy initializer, *Init_G*. In case the number of time intervals is larger than the number of projects, $|\mathcal{T}| \geq |\mathcal{P}|$, we can assume that the distribution resulting from *Init_G* is the best we can get for scheduling all projects. Otherwise, when the number of time intervals in the input exceeds the number of projects, $|\mathcal{T}| < |\mathcal{P}|$, a conflict between two projects occurs, which is inevitable. In such a case, at least two projects must be assigned to the same time interval. Thus, we can no longer assume that the greedy initial solution generator performs best without checking. To overcome this issue, we examine all other time intervals.

For each time interval $t' \neq t$, we check $IN\_COMMON_{(p,q)}\ \ \forall q \in SCHEDULE_{t'}$. If it does not exist (or is equal to an empty set), then we can schedule project p for the time interval t' instead of t. In other words, if p does not share an expert with any projects in a time interval, then it can be transferred to that time interval without any constraint violations. This transfer lets us analyze different conflicts between projects while still allowing experts to swap jobs among their teams. Typically, constraints would be violated when an expert gets overloaded with tasks or assigned to more than one skill in a single time interval. If the expert swap is within one project, the two experts are simply swapping tasks; since every project is allocated to only one time interval, no violations would be violated. If the expert swap is with an expert from outside the project, it is still a safe substitution since the choice is limited to the candidates available at that time. Thus, any expert swap in our proposed method uses the updated look-up tables and would be violation-free. This way, eventually, each project converges towards local optimality.

Evaluation Function:

Unlike the previous algorithm, the evaluation function in our proposed LNS method is simple. No constraints can be violated during the neighbourhood navigation; hence, it is safe to assume that the objective function alone can evaluate a solution. When exploring the feasible region, if $Obj(\mathcal{X}) > Obj(\mathcal{X}')$, then $\mathcal{X}$ is definitely a better state than $\mathcal{X}'$, and we allow the neighbourhood navigation.

Discussion:

Similar to the LS-Hybrid algorithm, we cannot prove optimality for LNS. In practice, LNS is applied until resource exhaustion and returns the current best solution after maximum idle iterations are passed. We get a random time interval t to explore, and then we pick a random project p among the ones assigned to the time interval t [5]. Inside project p, we try to find a better arrangement of experts that leads to a higher objective value. Integrating random choice of neighbours can be very robust in local search algorithms. A new neighbour found can be from any of the following relations.

1. Swapping two experts within one project by exchanging their tasks.

[5]The selected time interval must have at least one project assigned to it

2. Swapping an expert from one project with one from outside of the project.
   (a) Substituting an expert from one project with an unemployed expert.
   (b) Substituting an expert from one project with another expert who is in another team, but is available at this specific time interval.

As explained in the previous subsection, these steps are repeated if an alternative time interval t' is found to see if the time change is beneficial. The expert swap within the project would still result in the same objective function since the time interval is not considered in the objective function. Thus, we can omit re-checking the expert swaps within the project. Overall, five possible relations can produce a new neighbour with and without a time change.

Landing on a new neighbour seems more straightforward in LS-Hybrid than our proposed LNS based on the pseudocodes presented in 3.3.1 and 3.3.2. The variable permutation in LS-Hybrid covers all of the $\mathcal{E}$, $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{T}$ domains. However, in LNS, we have a much smaller loop to iterate through.

Let us analyze the computational cost of exploring a neighbourhood. To find a pair of candidates who can exchange their tasks within the selected time interval t and project p, we examine every pair of experts with different skills within (category 1) or outside (category 2) of p and determine if swapping them increases the objective function value. In either categories, there are $|TEAM_p|$ computations required for each project p to find the first swap candidate, where $|TEAM_p|$ is the number of experts assigned to project p. In category 1, the second swap candidate is selected from the same project p, so the computational cost of producing a new neighbour is $|TEAM_p|^2$, which is $\ll |\mathcal{E}|^2$. In category 2, the second swap candidate is selected from the list of experts not assigned to any project $q \neq p$ scheduled at time interval t. Therefore, the computational cost of producing a new neighbour is $|AVAILABLE_t| \times |TEAM_p|$, which is $< |\mathcal{E}|^2$. Moreover, to find a new time schedule t' for project p, we examine every time interval in $\mathcal{T}$. Thus, if the new neighbour is produced via a new schedule, the computational cost via category 2 is $\sum_{t' \in \mathcal{T}} |AVAILABLE_{t'}| \times |TEAM_p|$. Note that category 1 will not be considered anymore since it is redundant to examine experts within the project again.

Hence, the computational cost of exploring a neighbourhood is as below, which is of the order $\mathcal{O}(|\mathcal{T}| \times |\mathcal{E}|^2)$:

$$|TEAM_p|^2 + |AVAILABLE_t| \times |TEAM_p| + \sum_{t' \in \mathcal{T}} |AVAILABLE_{t'}| \times |TEAM_p|$$

.

Algorithm 3.3.2 LNS

```
Data: $\mathcal{E}, \mathcal{S}, \mathcal{P}, \mathcal{T}, PSR, Q, max\_iter$
Result: $W, V$
$iter\_counter \leftarrow max\_iter$
// initiate $V$ and $W$
$V, W, \text{SCHEDULE}, \text{TEAM}, \text{IN\_COMMON} \leftarrow Init\_G(\mathcal{E}, \mathcal{P}, \mathcal{S}, \mathcal{T}, PSR)$
$obj_{best} \leftarrow Obj(V, \mathcal{E}, \mathcal{S}, \mathcal{P}, Q)$ // set the best objective function
while $iter\_counter > 0$ do
    $t \leftarrow random(T)$ // pick a non-empty time interval $t$ randomly
    $p \leftarrow random(\text{SCHEDULE}[t])$ // pick a project scheduled at $t$ randomly
    // I) Investigate Swaps For The Current Time Schedule:
    $p\_skills \leftarrow [s \; \forall \; s \in \text{TEAM}[p]]$ // get all skills in project $p$
    for $sA\_i = 1$ to $len(p\_skills)$ do
        $sA \leftarrow p\_skills[sA\_i]$ // retrieve skill $sA$
        foreach $eA \in TEAM[p][sA]$ do
            // Category 1: choose the seocond expert from within $p$
            for $sB\_i = sA\_i + 1$ to $len(p\_skills)$ do
                $sB \leftarrow p\_skills[sB\_i]$ // retrieve skill $sB$
                foreach $eB \in TEAM[p][sB]$ do
                    $V_{nbr} \leftarrow SWAP(V, p, eA, eB, sA, sB)$
                    $obj_{nbr} \leftarrow Obj(V_{nbr}, \mathcal{E}, \mathcal{S}, \mathcal{P}, Q)$
                    if $obj_{nbr} > obj_{best}$ then
                        update $obj_{best}, V_{best}$
                        $swap_{best} \leftarrow (t, p, eA, eB, sA, sB)$
                        $new\_hire \leftarrow False$
                        $iter\_counter \leftarrow max\_iter$ // reset the idle counter
                    else
                        $iter\_counter \leftarrow iter\_counter - 1$ // decrement counter
            // Category 2: choose the second expert from outside of $p$
            NewHire($t$)
    // II) Investigate Swaps For Other Time Schedules:
    if $p$ is the only project in $t$ then
        foreach $t' \neq t \in \mathcal{T}$ do
            $new\_schedule \leftarrow True$
            $t_{new} \leftarrow t'$
            foreach $q \in SCHEDULE[t']$ do
                if $IN\_COMMON[(p, q)]$ then
                    // disqualify $t_{new}$, projects $p$ and $q$ share experts
                    $new\_schedule \leftarrow False$
        if $new\_schedule$ then
            $experts\_in\_p \leftarrow [e \ \forall \ e \in \mathrm{TEAM}[p]]$
            insert $experts\_in\_p$ into $\mathrm{AVAILABLE}_{nbr}[t]$
            remove $experts\_in\_p$ from $\mathrm{AVAILABLE}_{nbr}[t_{new}]$
            remove $p$ from $\mathrm{SCHEDULE}_{nbr}[t]$
            insert $p$ into $\mathrm{SCHEDULE}_{nbr}[t_{new}]$
            $W_{nbr}[(p, t)] \leftarrow 1 - W_{nbr}[(p, t_{new})]$
            // Category 2:  choose the second expert from outside of $p$
            NewHire($t_{new}$)
    NavigateToNeighbour()
```

```
Procedure NewHire
  input: time interval $t$
  foreach $eB \in AVAILABLE[t]$ do
      $V_{nbr} \leftarrow SWAP(V, p, eA, eB, sA)$
      $obj_{nbr} \leftarrow Obj(V, \mathcal{E}, \mathcal{S}, \mathcal{P}, Q)$
      if $obj_{nbr} > obj_{best}$ then
          update $obj_{best}, V_{best}$
          $swap_{best} \leftarrow (t, p, eA, eB, sA, sB)$
          $new\_hire \leftarrow True$
          reset $iter\_counter$
      else
          decrement $iter\_counter$
```

Procedure NavigateToNeighbour

```
if $swap_{best}$ and not $new\_schedule$ then
    update $V$
    $t, p, eA, eB, sA, sB \leftarrow swap_{best}$
    delete $eA$ from TEAM[$p$][$sA$]
    add $eB$ to TEAM[$p$][$sA$]
    // In case where the swap is within the project
    if $sB$ then
        delete $eB$ from TEAM[$p$][$sB$]
        add $eA$ to TEAM[$p$][$sB$]
    // In case where the swap is from outside of the project
    if $new\_hire$ then
        add $eA$ to AVAILABLE[$t$]
        remove $eB$ from AVAILABLE[$t$]
        $new\_hire \leftarrow False$
        update IN_COMMON
if $swap_{best}$ and $new\_schedule$ then
    update $V, W$
    update AVAILABLE, SCHEDULE
    $t, p, eA, eB, sA, sB \leftarrow swap_{best}$
    delete $eA$ from TEAM[$p$][$sA$]
    add $eB$ to TEAM[$p$][$sA$]
    add $eA$ to AVAILABLE[$t$]
    remove $eB$ from AVAILABLE[$t$]
    $new\_hire \leftarrow False$
    update IN_COMMON
delete $swap_{best}$
```

CHAPTER 4

Experiments

In this chapter, we present the results of the experiments to evaluate the performance of our proposed ILP model in Gurobi and the local search methods. Furthermore, we examine the effect of some parameters on execution time and memory usage.

4.1 Setup

We first provide details of the computational environment and the data sets employed, then present the results.

4.1.1 Environment

All experiments are conducted on a PC with an Intel(R) Core$^{\text{TM}}$ i5-8250U CPU @ 1.80 GHz and 8.00 GB memory. In the case of the local search methods, we examine 8 random initial states in parallel, as there are 8 CPU cores available. Python 3.8.1 is used for the exact and heuristic approaches. Gurobi 9.0 [11] was selected as the optimization solver for the proposed ILP model, and gurobipy, the Gurobi Python interface, was utilized for the implementation. For the algorithm performance evaluation, we measure the average time for ten executions of each algorithm, and for the study of simulation parameter effects, we take the average results of 10 random seed executions.

Other optimization solvers can be selected to solve the ILP model introduced in this thesis, such as CPLEX. We chose Gurobi for various reasons. Gurobi has a convenient Python interface, which allows the user to access different attributes in the solver or add a limitation to the MIP solver used by Gurobi. We use these features in our experiments to stop the optimization when a specific execution time or objective value has been reached. More importantly, Gurobi offers free academic licenses with no limits on model size and complete services. The one-year license can be renewed annually. However, other powerful tools such as CPLEX only offer a free solver for problems that have under 1,000 variables and 1,000 constraints [1].

In Gurobi's MIP solver, there is a valid lower bound at any time during the branch-and-bound search, sometimes called the best bound. Gurobi's documentation mentions that "this bound is obtained by taking the minimum of the optimal objective values of all of the current leaf nodes." The difference between the current upper and lower bounds is known as the **gap**, which demonstrates optimality when it is equal to zero.

4.1.2 Data

In order to evaluate the various parameters of our algorithms under a wide variety of conditions, we utilized synthetic data. For the most part, values were pulled from uniform random distributions. We occasionally introduced a bias for the quality measure to get cases with a higher chance of an expert not possessing a skill at all. The quality score Q_{es} is an arbitrary measure assessed for an expert in a specific skill. In our experiments, we bound Q_{es} to an integer between $\{0, 1, ..., 10\}$.

4.2 Case Study

To outline the objectives and desired outcomes, we generated a small-sized dataset using the described synthetic data, and then applied all three methods, ILP with Gurobi, LS-Hybrid, and LNS, without any limits on the execution time or the objective function. Consider the case study of twenty experts, two skills, four projects, and three time intervals with parameters shown in Tables 4.2.1, 4.2.2, and 4.2.3.

[1]According to ILOG CPLEX Optimization Studio pricing by the time this thesis document is written

Set	**Indexes**
$\mathcal{E}$	$\{1, 2, ..., 20\}$
$\mathcal{S}$	$\{1, 2\}$
$\mathcal{P}$	$\{1, 2, 3, 4\}$
$\mathcal{T}$	$\{1, 2, 3\}$

Table 4.2.1: **sample dsataset**

Project	**SQL**	**Python**
$P1$	1	2
$P2$	1	2
$P3$	1	1
$P4$	2	1

Table 4.2.2: $\boldsymbol{PSR}$, the number of experts needed for each skill in projects in the sample dataset

Expert	**SQL**	**Python**	**Expert**	**SQL**	**Python**
Brandon Crawford	9	0	Christa Morris	8	9
Eric Shae	0	7	Frank Huang	0	8
Gertrude Cook	6	0	Jack Carrozza	1	0
Jason Thier	0	7	Joyce Mills	0	4
Judith Castillo	0	0	Lois Tibbetts	3	6
Louise Anderson	5	3	Michael Adkinson	6	10
Nancy Bouie	0	1	Peter Towler	3	6
Raymond Dailey	0	6	Stan Bartlett	2	8
Stephan Hanson	4	0	Truman Holm	2	1
Verna Warren	0	6	Wanda Mckenney	5	3

Table 4.2.3: $\boldsymbol{Q}$, the skill level of each expert in the sample dataset

Expert	**Skill**	**Project**	**Time Interval**
Brandon Crawford	SQL	$P2$	$TI1$
Michael Adkinson	Python		
Christa Morris			
Brandon Crawford	SQL	$P1$	$TI2$
Michael Adkinson	Python		
Stan Bartlett			
Christa Morris	SQL	$P3$	
Frank Huang	Python		
Christa Morris	SQL	$P4$	$TI3$
Brandon Crawford			
Michael Adkinson	Python		

Table 4.2.4: **Gurobi**'s solution for the sample dataset
objective value: 98

Expert	**Skill**	**Project**	**Time Interval**
Christa Morris	SQL	$P2$	$TI1$
Gertrude Cook	Python		
Raymond Dailey			
Gertrude Cook	SQL	$P4$	$TI2$
Stan Bartlett			
Michael Adkinson	Python		
Louise Anderson	SQL	$P1$	$TI3$
Michael Adkinson	Python		
Christa Morris			
Gertrude Cook	SQL	$P3$	
Jason Thier	Python		

Table 4.2.5: **LS-Hybrid**'s solution for the sample dataset
objective value: 73

Expert	**Skill**	**Project**	**Time Interval**
Christa Morris	SQL	$P1$	$TI1$
Michael Adkinson Frank Huang	Python		
Gertrude Cook Brandon Crawford	SQL	$P4$	
Stan Bartlett	Python		
Brandon Crawford	SQL	$P2$	$TI2$
Michael Adkinson Christa Morris	Python		
Brandon Crawford	SQL	$P3$	$TI3$
Michael Adkinson	Python		

Table 4.2.6: **LNS**'s solution for the sample dataset
objective value: 96

The objective values of the solutions found by Gurobi, LS-Hybrid, and LNS are **98**, **73**, and **96**, respectively. The teams selected by each method are shown in Tables 4.2.4, 4.2.5, and 4.2.6. According to the sample dataset, experts may have an inadequate or zero quality level in some skills. None of the methods choose an expert with a zero quality score when other options are available. Also, since an expert cannot work on two tasks assigned at the same time interval, all three methods find a minimum number of conflicts among the projects. This way, there is a broader selection of available experts for each task.

As reported by Gurobi, the globally optimum solution occurs where project $P1$ and $P3$ are in conflict. LS-Hybrid adopts the same conflict configuration but reaches a local optimum it cannot avoid due to its small search step. Although the LNS method establishes the conflict differently in its final solution, it obtains a locally optimal solution with nearly the same objective value as the globally optimal solution, greatly outperforming LS-Hybrid.

4.3 Experimental Evaluation

In this section, we evaluate the performance of the ILP, LS-Hybrid, and LNS methods on larger datasets. Also, we examine the effect of the size of the input parameters on the number of constraints produced. We aim to answer the following questions:

- **Q1: Parameter Effect Analysis.** What is the effect of the size of the input parameters, $\mathcal{E}$, $\mathcal{S}$, $\mathcal{P}$, and $\mathcal{T}$, on the number of the constraints?
- **Q2: Objective Comparison.** How close are the results of the heuristic algorithms to the global optimal solution in fixed execution times?
- **Q3: Comparative Performance.** How do ILP, LS-Hybrid and LNS compare in terms of execution time to reach the same objective value?

4.3.1 Q1: Parameter Effect Analysis

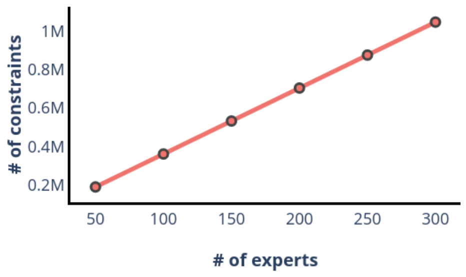
In order to evaluate and compare the performance of both Gurobi and the local search-based methods, we first need to examine the effect that different size parameters have on the problem and solution. We can achieve this by examining the number of individual constraints that are produced for datasets with different numbers of experts, skills, projects, time instances as well as average number of experts per skill.

$\|\mathcal{E}\|$	**# of Constraints**	$\|\mathcal{S}\|$	**# of Constraints**
50	189180	1	77140
100	361180	2	248160
150	533180	3	533180
200	705180	4	932200
250	877180		
300	1049180		
$\|\mathcal{P}\|$	**# of Constraints**	$\|\mathcal{T}\|$	**# of Constraints**
10	127090	5	521780
15	295260	10	527480
20	533180	15	533180
25	840850	20	538880
30	1218270	25	544580

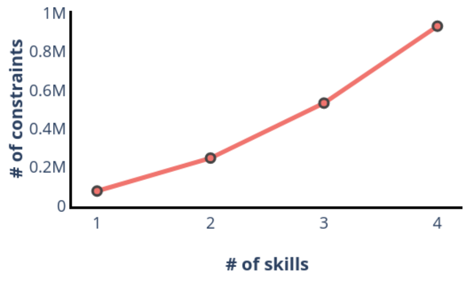
Table 4.3.1: **Effect of the change in parameter sizes on the number of constraints**

To get a better insight into the effect of each data entry on the number of constraints, we refer to the mathematical model proposed for the problem in section 3.2 of Chapter 3. To analyze each parameter, we fix all but one of the $|\mathcal{E}| = 150$, $|\mathcal{S}| = 3$, $|\mathcal{P}| = 20$, $|\mathcal{T}| = 15$, and $|PSR_{max}| = 2$ values, while varying the remaining quantity. The number of constraints produced can be used to determine the size and/or difficulty of solving the particular ILP problem. The results for each quantity can be seen in Table 4.3.1. Note that the maximum PSR depends on the number of experts in the input. Increasing the maximum PSR allowed in projects can lead to expert shortage and the lack of any feasible solution. Also, increasing or decreasing the maximum value allowed for PSR does not affect the number of constraints produced in a problem. For instance, by increasing the maximum PSR from 1 to 4 for the same setup, we get the flat number of 533180 constraints produced.

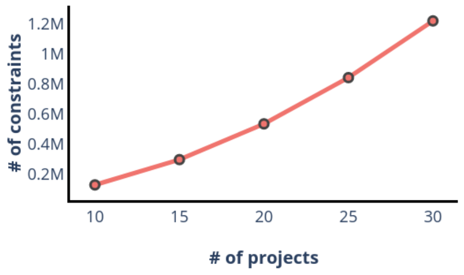
As derived from the mathematical model 3.2, we plot each part of the data from Table 4.3.1 into four figures shown in Figure 4.3.1a for better observation. The number of constraints produced in the problem grows linearly by increasing the number of experts. Similarly, on a smaller scale based on their y-axis, increasing the number of time intervals shown in Figure 4.3.1d raises the number of constraints in a linear fashion. Moreover, Increasing the number of skills above four leads to an infeasible solution, and the problem becomes unsolvable. Therefore, we can see that the maximum possible number of skills highly depends on the number of experts and projects. Lastly, we observe the maximum number of constraints, 1218270, produced when increasing the number of projects with nonlinear growth.



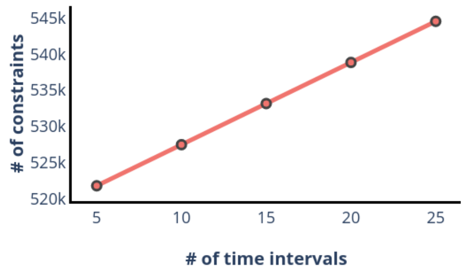
(a) $|\mathcal{S}| = 3$, $|\mathcal{P}| = 20$, $|\mathcal{T}| = 15$



(b) $|\mathcal{E}| = 150$, $|\mathcal{P}| = 20$, $|\mathcal{T}| = 15$



(c) $|\mathcal{E}| = 150$, $|\mathcal{S}| = 3$, $|\mathcal{T}| = 15$



(d) $|\mathcal{E}| = 150$, $|\mathcal{S}| = 3$, $|\mathcal{P}| = 20$

Fig. 4.3.1: Effect of the number of experts (a), skills (b), projects (c), time intervals (d), and maximum experts per skill (e) on the number of constraints generated.

Therefore, for all upcoming experiments, we examine how different metrics scale in relation to the number of projects $|\mathcal{P}|$. It displays the exponential nature of the problem while allowing for valid values without needing to modify any of the other parameters.

4.3.2 Q2: Objective Comparison.

LS-Hybrid and LNS find locally optimal objectives and do not guarantee to provide the globally optimal solution. Thus, we fix a maximum execution time to evaluate the solutions' objective values fairly. By setting the Time_Limit parameter on Gurobi and manually computing the execution time on LNS, both methods will terminate when the execution time is exceeded. The results of this experiment can be seen in Fig 4.3.2. As mentioned before, LS-Hybrid searches both infeasible and feasible regions; thus, its execution time is sizable compared to the other two methods. Most of the solutions found by LS-Hybrid under the limited execution time were infeasible, so we do not consider it for this experiment. Gurobi surpasses LNS when the number of projects is less than the number of time intervals. As we increase $|\mathcal{P}|$, Gurobi's execution time increases, resulting in premature termination of the optimization and some performance loss. Therefore, although Gurobi performs $\times 2.5$ better than LNS for $|\mathcal{P}| = 10$, it performs equally or worse than LNS for $|\mathcal{P}| \geq 45$

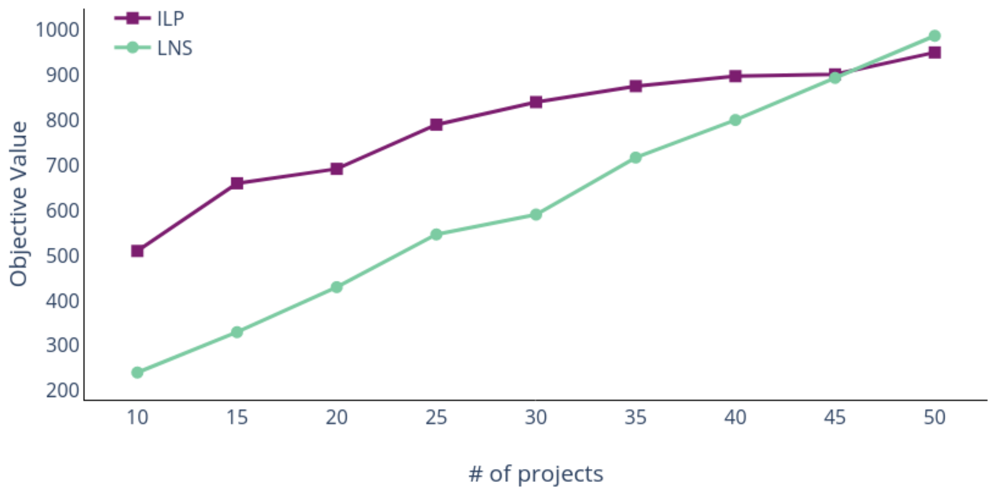


Fig. 4.3.2: Objective value comparison of Gurobi and LNS.
$|\mathcal{E}| = 600$, $|\mathcal{S}| = 3$, $|\mathcal{T}| = 20$, $|PSR_{max}| = 2$

4.3.3 Q3: Comparative Performance.

In order to explore the comparison of execution time among all three methods, we set an objective goal for all methods. To prepare for the experiment, we employ Gurobi without limitations to obtain the globally optimal solution. Then, we alter Gurobi's best objective value parameter, BestObjStop, so that the objective function reaches at least 60% of the previously calculated optimal one. Next, we attempt to reach at least the same objective value with LS-Hybrid and LNS. We examine the execution times for different numbers of projects. Figure 4.3.3 presents the results. Another way to set termination criteria for Gurobi's MIP solver is to fix the gap between the lower and upper objective bound, MIPGapAbs, in a way that the optimization terminates when reaching a MIP gap of 60%.

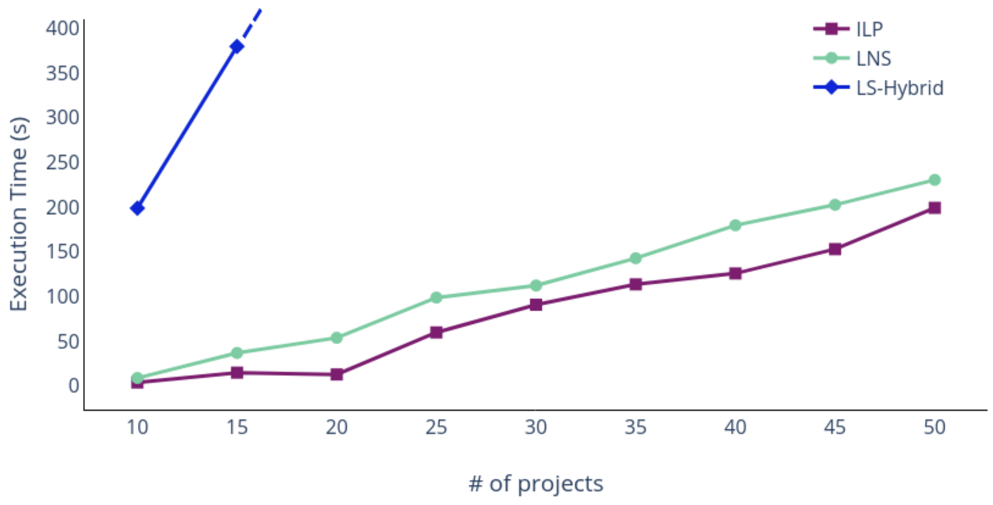


Fig. 4.3.3: Execution time performance comparison of ILP, LS-Hybrid, and LNS. $|\mathcal{E}| = 600$, $|\mathcal{S}| = 3$, $|\mathcal{T}| = 20$, $|PSR_{max}| = 2$

As can be seen, although LS-Hybrid finds a local optimum, it has a high execution time due to its considerable number of neighbour traversals. The calculation for $|\mathcal{P}| > 20$ was prohibitively expensive. The execution time for Gurobi and LNS roughly grows equally. Gurobi's overall execution time for the first datasets is better than LNS when no limitation is applied. However, the computational cost of Gurobi grows exponentially with respect to the number of projects $|\mathcal{P}|$. As seen in Figure 4.3.3, Gurobi performs about the same as LNS when the objective value is restricted.

It should also be noted that the space required by Gurobi is proportional to the number of constraints. Thus, it grows exponentially with the number of projects as shown in the Figure 4.3.1c in the previous subsection. In contrast, the space required for LS-Hybrid and LNS grows quadratically with the size of the dataset. See Chapter 3.3.2 for more details.

CHAPTER 5

Conclusions and Future Steps

5.1 Conclusions

In this work, we studied a team formation problem where we are given a set of projects, each requiring specific skills that must be completed in a particular time. Additionally, we are given a set of candidates where each individual has a skill set alongside a quality score associated with each one. We explored various techniques to achieve a global or local solution for finding the most highly qualified teams of experts for all projects.

We modelled the problem using integer linear programming and examined it via Gurobi, the commercial optimizer. Then we explored two heuristic approaches to tackle the proposed TFP, LS-Hybrid and LNS. Due to heuristic algorithm characteristics, we cannot prove our solution's proximity to the optimal. However, the exact algorithms, such as the ones used in Gurobi to solve the ILP model, achieve a globally optimal solution. We did an approximation accuracy analysis and showed that compared to the results from ILP, our proposed LNS's accuracy is between 75-90%.

An advantage of heuristic approaches, like the ones used in this thesis, over constructive methods is that the algorithm can be stopped prematurely at any time during the execution to get a solution. That solution may or may not be locally optimal and most likely will not be globally optimal, but it is still a complete solution to the problem. There is, therefore, an inherent trade-off between accuracy and time between the optimization and local search approaches, respectively.

Furthermore, we must mention that the number of constraints produced when using ILP can lead to a considerable amount of memory with particularly large test cases. On the other hand, in both heuristic approaches, the memory is limited to only the current assignments of V and W and the defined data structures.

The LS-Hybrid solution provided in this thesis is not problem specific and can be adopted by other similar problems. However, some criticism is directed at the practicality of classic TFP [13].

One of the main assumptions in most TFPs is that the experts are assumed to be idle, which is somewhat unrealistic. In real-world scenarios, experts can get hired by another company/institution by the time we come to a decision. On the other hand, we believe that it is beneficial for a company with multiple projects to utilize this model to consider schedules, acknowledge costs (such as setting annual budgets), and estimate the number of candidates they need to hire depending on the optimization results.

Moreover, our proposed methods can be a tool to analyze what-if scenarios during resource planning. Experiments similar to the ones shown in Chapter 4 can be employed to determine how much the overall project gains are affected by different parameters.

5.2 Future work

There are other possibly efficient techniques to linearize quadratic constraints, similar to the one mentioned in Chapter 3.2 [22]. In the same chapter, our linear alternative is suggested in equations (3, 4, and 5).

This research can be further extended in several ways. To examine other possible neighbourhoods for the LNS, we suggest unassigning all experts to a single time interval and solving the smaller subproblem. It would allow the expert to swap jobs not only within a project but also between multiple projects that are assigned to the same time interval.

Moreover, it may be interesting to evaluate the algorithms developed in various real-world datasets, such as data extracted from GitHub repositories or data scraped from platforms like LinkedIn.

APPENDIX A

Implementation

A.1 ILP Optimization Using Gurobi

```
def ilp(experts, projects, skills, timeslots, Q, PSR):
    num_experts, num_projects, num_skills, num_timeslots = len(experts
        ↪ ), len(projects), len(skills), len(timeslots)

    # -- Create a Model:
    mdl = gp.Model("Maximum Expert Quality Hiring Problem")

    ##############################################
    # Variables #
    ##############################################
    # -- Wpt: 1 if project p is selected at timeslot t; 0 otherwise
    W = mdl.addVars(projects, timeslots, vtype=gp.GRB.BINARY, name="W
        ↪ ")

    # -- Vesp: 1 if expert e is assigned to skill s in project p; 0
        ↪ otherwise
    V = mdl.addVars(experts, skills, projects, vtype=gp.GRB.BINARY,
        ↪ name="V")
```

```
# -- Zpqt: 1 if project p and q are assigned to the same timeslot
    ↪ t; 0 otherwise
Z = mdl.addVars(projects, projects, timeslots, vtype=gp.GRB.BINARY
    ↪ , name="Z")

mdl.update()

##############################################
# Mathematical Formulation #
##############################################
# -- Objective: maximize the quality of the selected experts
mdl.setObjective(gp.quicksum(Q[(e, s)] * V[(e, s, p)] for e in
    ↪ experts for s in skills for p in projects),
               gp.GRB.MAXIMIZE)

# Constraints:
# -- Constraint 1: Expert e cannot be assigned to skills in
    ↪ different projects at the same timeslot
mdl.addConstrs((V[e, s, q] + V[e, u, p] + gp.quicksum(Z[p, q, t]
    ↪ for t in timeslots) <= 2
              for e in experts for s in skills for u in skills for
                  ↪  p in projects for q in projects if p != q),
             name="Constraint1")

# -- Constraint 2-4: Controlling conflicts between projects
mdl.addConstrs((Z[p, q, t] >= W[p, t] + W[q, t] - 1
              for p in projects for q in projects if p != q for t
                  ↪ in timeslots),
             name="Constraint2")
```

```
mdl.addConstrs((Z[p, q, t] <= W[p, t]
               for p in projects for q in projects if p != q for t
                   ↪ in timeslots),
              name="Constraint3")

mdl.addConstrs((Z[p, q, t] <= W[q, t]
               for p in projects for q in projects if p != q for t
                   ↪ in timeslots),
              name="Constraint4")

# -- Constraint 5 : All projects must be selected
mdl.addConstrs((gp.quicksum(W[p, t] for t in timeslots) == 1 for p
    ↪  in projects),
              name="Constraint5")

# -- Constraint 6: Enough experts must be assigned to skill s in
    ↪ project p.
mdl.addConstrs((gp.quicksum(V[e, s, p] for e in experts) <= PSR[s,
    ↪  p] for s in skills for p in projects),
              name="Constraint6")

# -- Constraint 7: A person should be allowed to one thing at the
    ↪ time
mdl.addConstrs((gp.quicksum(V[e, s, p] for s in skills) <= 1 for e
    ↪  in experts for p in projects),
              name="Constraint7")

# Update the model
mdl.update()
```

```
##############################################
# Optimization #
##############################################
# Make a path for the data
PATH = '_'.join([str(num_experts) + 'e', str(num_skills) + 's',
               str(num_projects) + 'p', str(num_timeslots) + 't'])
os.makedirs(PATH, exist_ok=True)
mdl.write(PATH + "/model.lp")

# Optimize the model
mdl.optimize()
print("MODEL RUNTIME: ", mdl.Runtime)
```

APPENDIX B

LS Algorithms' Initial Solution

In this chapter, we elaborate on the two initial solution generators used in Algorithms 3.3.1 and 3.3.2. We discuss the *Init_R* function followed by the *Init_G*. Consider the example below with five experts, two skills, four projects, and three time intervals. We will use this example in the next sections for better interpretation.

Set	Indexes
$\mathcal{E}$	$\{0, 1, 2, ..., 10\}$
$\mathcal{S}$	$\{0, 1\}$
$\mathcal{P}$	$\{0, 1, 2, 3\}$
$\mathcal{T}$	$\{0, 1, 2\}$

Table B.0.1: **Example 1.** with some PSR, and Q

B.1 Purely Random Initial Solution (*Init_R*)

For variables V, we iterate through the set of experts, skills, and projects, then assign a random binary digit to any of the variables. The results for Example 1. shown in Table B.0.1 could be such as below, which is not necessarily a feasible solution:

```
# The first rows of the dictionary are shown for clarification
# Definition: V[(e, s, p)] = 0 or 1
V ={(e0, s0, p0): 0, (e0, s0, p1): 1, (e0, s0, p2): 0, (e0, s0, p3): 0,
    (e0, s1, p0): 0,
                                                    ..., (e10, 1, 3): 1}
```

For variables W, we iterate through the set of projects, but we only allow one time-interval selection for each. The reason will be explored and explained more in detail in the next subsection. Again, for Example 1. the initializer could result in:

```
# Each row is related to one project and has only one assignment.
# Multiple project assignments to one time interval is possible,
# such as p1 and p2 to t1.
# Definition: W[(s, p)] = 0 or 1
W ={(s0, p0): 1, (s0, p1): 0, (s0, p2): 0,
    (s1, p0): 0, (s1, p1): 1, (s1, p2): 0,
    (s2, p0): 0, (s2, p1): 1, (s2, p2): 0,
    (s3, p0): 0, (s3, p1): 0, (s3, p2): 1}
```

We keep the indexes to the selected time interval and their associated projects in the data structure, *SCHEDULE* for faster access during the local search. To get a better conception, look at the following example:

```
# This is a lookup table, not a part of the solution,
# which keys time intervals by their selected projects.
# Definition: SCHEDULE[p] = t
SCHEDULE ={p0: t0,
           p1: t1,
           p2: t1,
           p3: t2}
```

B.2 Greedy Initial Solution (*Init_G*)

The generated solution via a greedy approach will still be randomly assigned; the only difference is that it is also a feasible solution. Experts can be assigned to skills they do not possess, where they have a quality score Q equal to zero. The constructed solution possibly has an undesirable objective value, but at least all the assignments are guaranteed to be feasible. The two data structures generated alongside the solution are as follows:

In the altered *SCHEDULE* data structure, we key the list of projects by the time interval they are scheduled. Below is an example of the new *SCHEDULE*:

```
# This is a lookup table, not a part of the solution,
# which keys the list of projects by their selected time interval.
# Definition: SCHEDULE[t] = [p for p in P if W[(p,t) == 1]
SCHEDULE ={t0: [p0],
           t1: [p1, p2],
           t2: [p3]}
```

TEAM is also an additional data structure that is generated to keep track of task assignments to experts. It shows the assignment of the set of experts for each skill in a project. TEAM[p][s] contains the set of all experts hired for project p, skill s. In a general sense, the team formed for each project is held within TEAM[p]. For a better understanding, look at the following example:

```
# This is a lookup table, not a part of the solution,
# For each project key, another dictionary keeps the experts' information for each
                                                ↪  skill in the project.
# Definition: TEAM[p] = {s: [e]}
TEAM ={p0: {s0: [e0, e1]},
       p1: {s1: [e0, e4]},
       p2: {s0: [e1], s1: [e2, e3]},
       p3: {s0:[e1], s1:[e2, e5]}}
```

AVAILABLE is a data structure consisting of all the experts that are available to be hired at specific time intervals. Keeping lists of available experts at each time interval accelerates and organizes the neighbourhood navigation for the large neighbourhood search. Below is an example for better understanding. Assuming the current formed teams are selected as in the *TEAMS* above, expert $e5, e6$, and the rest are not assigned to any projects yet, so they are idle or unemployed at all time intervals. Also, due to the current scheduling, some experts hired for other projects might still be available at a given time.

```
# This is a lookup table, not a part of the solution,
# Lists of experts that are available at a given time interval.
# Definition: AVAILABLE[t] = [e]
AVAILABLE ={t0: [e2, e3, e4, e5, e6, e7, e8, e9, e10],
            t1: [e5, e6, e7, e8, e9, e10],
            t2: [e0, e3, e4, e6, e7, e8, e9, e10],}
```

Lastly, *IN_COMMON* is a graph that indicates whether or not two projects have hired any experts in their assignments in common. This data structure is a lookup table that speeds up the queries needed in the large neighbourhood search. The edge value can be the set of all experts in common between two projects or just a binary value indicating the existence of any mutual experts. For clarification, look at the following example:

```
# This is a lookup table, not a part of the solution,
# A graph where nodes are projects and edges show what experts each two projects
                                            ↪ have in common.
# Definition: IN\_COMMON[(p, q)] = [e]
IN_COMMON ={(p0, p1) :[e0],
            (p0, p2) :[e1],
            (p0, p3) :[e1],
            (p1, p2) :[],
            (p1, p3) :[],
            (p2, p3) :[e1, e2]}
```

REFERENCES

[1] Abdel-Basset, M., Abdel-Fatah, L., and Sangaiah, A. K. (2018). Metaheuristic algorithms: A comprehensive review. *Computational intelligence for multimedia big data on the cloud with engineering applications*, pages 185–231.

[2] Ahuja, R. K., Ergun, Ö., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102.

[3] Allen, N. J. and Hecht, T. D. (2004). The 'romance of teams': Toward an understanding of its psychological underpinnings and implications. *Journal of Occupational and Organizational Psychology*, 77(4):439–461.

[4] Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., and Leonardi, S. (2010). Power in unity: forming teams in large-scale community systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 599–608.

[5] Baykasoglu, A., Dereli, T., and Das, S. (2007). Project team selection using fuzzy optimization approach. *Cybernetics and Systems: An International Journal*, 38(2):155–185.

[6] Chen, S.-J. and Lin, L. (2004). Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE transactions on Engineering Management*, 51(2):111–124.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition.

[8] Eren, Y., İbrahim B. Küçükdemiral, and İlker Üstoğlu (2017). Chapter 2 - introduction to optimization. In Erdinç, O., editor, *Optimization in Renewable Energy Systems*, pages 27–74. Butterworth-Heinemann, Boston.

[9] Fitzpatrick, E. L. and Askin, R. G. (2005). Forming effective worker teams with multi-functional skill requirements. *Computers &amp; Industrial Engineering*, 48(3):593–608.

[Glassdoor] Glassdoor. Glassdoor economic research.

[11] Gurobi Optimization, LLC (2022). Gurobi Optimizer Reference Manual.

[12] Jaddi, N. S. and Abdullah, S. (2020). Global search in single-solution-based metaheuristics. *Data Technologies and Applications.*

[13] Juárez, J., Santos, C. P., and Brizuela, C. A. (2022). A comprehensive review and a taxonomy proposal of team formation problems. *ACM Computing Surveys*, 54(7):1–33.

[14] Katzenbach, J. R. and Smith, D. K. (1993). The discipline of teams. *Harvard Business Review.*

[15] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[16] Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

[17] Lappas, T., Liu, K., and Terzi, E. (2009). Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 467–476.

[18] Lawler, E. L. and Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719.

[19] Majumder, A., Datta, S., and Naidu, K. (2012). Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1005–1013.

[20] Mathews, G. B. (1896). On the Partition of Numbers. *Proceedings of the London Mathematical Society*, s1-28(1):486–490.

[21] Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

[22] Oral, M. and Kettani, O. (1992). A linearization procedure for quadratic and cubic mixed-integer problems. *Operations Research*, 40(1-supplement-1):S109–S116.

[23] Palpant, M., Artigues, C., and Michelon, P. (2004). Lssper: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1):237–257.

[24] Selvi, S. T., Baskar, S., and Rajasekar, S. (2018). An intelligent approach based on metaheuristic for generator maintenance scheduling. In *Classical and recent aspects of power system optimization*, pages 99–136. Elsevier.

[25] Souza Brito, S., Gambini Santos, H., and Miranda Santos, B. H. (2014). A local search approach for binary programming: Feasibility search. In *International Workshop on Hybrid Metaheuristics*, pages 45–55. Springer.

[26] Sukthankar, G. and Sycara, K. (2006). Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In *AAAI*, volume 6, pages 716–721.

[27] Taccari, L. (2016). Integer programming formulations for the elementary shortest path problem. *European Journal of Operational Research*, 252(1):122–130.

[28] Talbi, E. and Metaheuristics, G. (2009). From design to implementation. vol. 74. hoboken.

[29] van de Water, T., van de Water, H., and Bukman, C. (2007). A balanced team generating model. *European Journal of Operational Research*, 180(2):885–906.

[30] Wi, H., Oh, S., Mun, J., and Jung, M. (2009). A team formation model based on knowledge and collaboration. *Expert Systems with Applications*, 36(5):9121–9134.

[31] Yang, X.-S. and Deb, S. (2014). Cuckoo search: recent advances and applications. *Neural Computing and applications*, 24(1):169–174.

[32] Zzkarian, A. and Kusiak, A. (1999). Forming teams: an analytical approach. *IIE transactions*, 31(1):85–97.

VITA AUCTORIS

NAME:	Yalda Yazdanpanah
PLACE OF BIRTH:	Tehran, Iran
YEAR OF BIRTH:	1995
EDUCATION:	
	Shiraz University, B.Sc in Computer Engineering, Shiraz, Iran, 2018
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2022