Mathematical Model for TalentBridge Connect Team Selection Problem

January 2, 2026

Contents

Chapter 1

Problem Definition

The TalentBridge Connect Team Selection Problem is a **Weighted Set Cover Problem with Proficiency Constraints**, which is NP-hard.

1.1 Weighted Set Cover

Theorem 1. *The Weighted Set Cover Problem with Proficiency Constraints can solve the Set Cover problem [3].*

Proof. Consider an instance $I = (U, \mathcal{C})$ of the Set Cover problem, where:
- $U = \{e_1, e_2, \ldots, e_n\}$ is the universe of elements,
- $\mathcal{C} = \{S_1, S_2, \ldots, S_m\}$ is a collection of subsets of U.

We construct an instance I' of the Weighted Set Cover with Proficiency Constraints as follows:

1. Let the set of required skills be $\mathbf{S} = U$.
2. For each skill $e_i \in \mathbf{S}$, set the required proficiency level $r_i = 1$.
3. For each set $S_j \in \mathcal{C}$, create a freelancer f_j such that:
   - f_j possesses exactly the skills in S_j.
   - For each skill $s \in S_j$, set f_j's proficiency in s to 1.
4. Set the weight (cost) of each freelancer f_j to 1.

Now, observe that in I':

- A team of freelancers $T \subseteq \{f_1, f_2, \ldots, f_m\}$ covers a skill e_i if and only if at least one freelancer in T has e_i in their skill set (with proficiency 1, meeting the required proficiency $r_i = 1$).

- Thus, T covers all skills in $\mathbf{S}$ if and only if the union of the corresponding sets S_j (for $f_j \in T$) equals U.

- The total weight of T is exactly $|T|$, the number of freelancers selected.

Therefore, a minimum-weight team in I' corresponds to a minimum set cover in I. Since the transformation is polynomial-time (in fact, linear in the size of I), the Weighted Set Cover with Proficiency Constraints can solve the Set Cover problem. □

Theorem 1. *The Weighted Set Cover Problem with Proficiency Constraints is NP-hard.*

Proof. Since Set Cover is NP-hard [2], and the above reduction shows it is a special case of Weighted Set Cover with Proficiency Constraints (with all proficiencies and weights set to 1), the latter is also NP-hard. □

1.2 Formal Input Model

Let:

- $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$ be the set of n required skills
- For each skill $s_i \in S$, let $r_i \in \mathbb{Z}^+$ be the required proficiency level
- $\mathbf{F} = \{f_1, f_2, \ldots, f_m\}$ be the set of m available freelancers

For each freelancer $f_j \in F$:

- $w_j \in \mathbb{R}^+$ is the hourly wage (cost)
- $\mathbf{T_j} \subseteq S$ is the subset of skills freelancer f_j possesses
- For each skill $s \in T_j$, let $c_{js} \in \mathbb{Z}^+$ be the competency level

1.3 Constraints

1.3.1 Coverage Constraint

For every skill $s_i \in S$, there must exist at least one freelancer f_j in the selected team such that:

- $s_i \in T_j$ (freelancer has the skill)
- $c_{ji} \geq r_i$ (freelancer's competency meets or exceeds required level)

Mathematically:

$$\forall s_i \in S, \exists f_j \in F' \text{ such that: } s_i \in T_j \wedge c_{ji} \geq r_i$$

where $F' \subseteq F$ is the selected team.

1.4 Objective Function

Minimize the total cost:

$$\text{minimize} \sum_{f_j \in F'} w_j$$

1.5 Decision Variables

Let $x_j \in \{0, 1\}$ be binary variables indicating whether freelancer f_j is selected:

$$x_j = \begin{cases} 1 & \text{if freelancer } f_j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

1.6 Integer Linear Programming Formulation

$$\begin{aligned} \text{minimize} \quad & \sum_j w_j \cdot x_j \\ \text{subject to} \quad & \sum_{\substack{j: s_i \in T_j \\ c_{ji} \geq r_i}} x_j \geq 1, \quad \forall s_i \in S \\ & x_j \in \{0, 1\}, \quad \forall f_j \in F \end{aligned}$$

1.7 Bitmask Representation

Define for each freelancer f_j a **coverage mask** $M_j \in \{0, 1\}^n$:

$$M_j[i] = \begin{cases} 1 & \text{if } s_i \in T_j \wedge c_{ji} \geq r_i \\ 0 & \text{otherwise} \end{cases}$$

The problem becomes: find the subset $F' \subseteq F$ that minimizes $\sum_{j \in F'} w_j$ such that:

$$\bigvee_{j \in F'} M_j = \{1\}^n \quad \text{(bitwise OR of all selected masks equals full coverage)}$$

1.8 Output Specification

The solution is a tuple (C^*, F^*):

- $C^* \in \mathbb{R}^+$: minimum total cost (sum of wages)
- $F^* \subseteq F$: optimal team of freelancers

If no team can cover all requirements, return $(\infty, \emptyset)$.

1.9 Computational Complexity

1.9.1 Complexity Classification

Theorem: This problem is **NP-hard**.
Proof:

- When all competency levels are 1–that is the candidate is either competent enough in the skill required or he is not–, the problem reduces to the **Weighted Set Cover Problem**, which is known to be NP-hard.
- The general case with competency levels is at least as hard.

1.9.2 Time Complexity Analysis

- **Brute Force**: $O(2^m \cdot m \cdot n)$ - check all subsets of freelancers
- **Dynamic Programming with Bitmask**: $O(m \cdot 2^n)$ - feasible when $n \leq 20$
- **Approximation**: $O(m \cdot n)$ for greedy $(\ln n)$-approximation algorithms

1.9.3 Space Complexity

- **DP Solution**: $O(2^n)$ for storing intermediate states

1.10 Problem Classification

- **Type**: Combinatorial Optimization
- **Class**: NP-hard
- **Special Cases**:
  - **Polynomial**: When each freelancer covers exactly one skill $\rightarrow$ Assignment Problem
  - **Polynomial**: When m is small $\rightarrow$ Brute force feasible
  - **Polynomial**: When n is small $\rightarrow$ DP with bitmask feasible

1.11 Practical Considerations

For real-world instances where $n > 20$, we would need:

1. **Approximation algorithms** (greedy, LP rounding)
2. **Heuristic methods** (genetic algorithms, local search)
3. **Commercial integer programming solvers**
4. **Problem decomposition** techniques

Chapter 2

Algorithmic Analysis and Complexity Proofs

2.1 Polynomial-Time Reduction

We first prove that the reduction from the original problem with m freelancers to the bitmask representation has polynomial time complexity and produces an instance where the state space is bounded.

Theorem 2 (Reduction Complexity). *The preprocessing step that converts m freelancers to relevant skill masks runs in $O(m \cdot n)$ time and produces at most $\min(2^n, m)$ distinct masks.*

Proof. For each freelancer f_j, we check n skills, performing constant-time operations for each skill check and bitmask update. The total operations are:

$$\sum_{j=1}^{m} O(n) = O(m \cdot n)$$

The number of distinct masks is trivially bounded by 2^n (all possible skill combinations) and also by m (number of input freelancers), giving $\min(2^n, m)$. □

2.1.1 Brute-Force Algorithm Analysis

Theorem 3 (Brute-Force Complexity). *The brute-force algorithm has time complexity $O(\min(2^n, m) \cdot 2^{\min(2^n,m)})$ and space complexity $O(\min(2^n, m))$.*

Proof. Let $k = \min(2^n, m)$ be the number of distinct masks after reduction. The algorithm:

1. Generates all subsets of masks: $\sum_{i=1}^{k} \binom{k}{i} = 2^k$ subsets
2. For each subset, checks coverage: $O(n)$ operations
3. For valid covers, computes cost: $O(k)$ operations

Total time: $O(2^k \cdot (n+k)) = O(2^k \cdot k)$ since $k \geq n$ in non-trivial cases.

Space complexity is dominated by storing k masks and their optimal freelancers. □

2.1.2 Dynamic Programming Algorithm Analysis

Theorem 4 (DP Algorithm Complexity)**.** *The dynamic programming algorithm has time complexity $O(m \cdot n + k \cdot 2^n)$ and space complexity $O(2^n)$, where $k = \min(2^n, m)$.*

Proof. The algorithm consists of:

- **Reduction phase**: $O(m \cdot n)$ as established

- **DP initialization**: $O(2^n)$ for initializing the DP table (it could be O(1) if we simply *malloc* to reseve memory but the array would be full of garbage. This, of course, assuming the memory allocation is independent of the input; which is false, but useful)

- **DP computation**: For each of k masks, iterate through 2^n states: $O(k \cdot 2^n)$

Total time: $O(m \cdot n + 2^n + k \cdot 2^n) = O(m \cdot n + k \cdot 2^n)$

Space is dominated by the DP table of size 2^n. □

2.1.3 Optimality Guarantees

Theorem 5 (Solution Optimality)**.** *Both algorithms guarantee finding the optimal solution when one exists.*

Proof. **Brute-force**: Examines all possible team combinations, guaranteeing optimality.

DP algorithm: Uses dynamic programming with complete state representation. The DP transition:

$$dp[new\_mask] = \min(dp[new\_mask], dp[current\_mask] + cost(mask))$$

systematically explores all possible ways to achieve each skill coverage state, guaranteeing the optimal solution is found. □

2.1.4 Empirical Performance Comparison

The theoretical analysis is supported by empirical observations:

- **Small** n **(≤ 10)**: Both algorithms perform well, with DP showing better scaling with m
- **Medium** n **($11 - 20$)**: DP algorithm remains feasible while brute-force becomes impractical
- **Large** n **(> 20)**: Both algorithms face challenges, necessitating approximation approaches

This analysis justifies our algorithmic approach and demonstrates that the exponential factor is well-controlled in practical deployment scenarios.

2.2 Greedy Approximation Algorithm

2.2.1 Greedy Algorithm Description

The greedy algorithm [3] for the team selection problem iteratively selects the most cost-effective freelancer at each step, where cost-effectiveness is defined as the ratio of wage to the number of newly covered skills.

Algorithm 1 Greedy Team Selection Algorithm

Require: Project requirements, list of freelancers
Ensure: Team of freelancers covering all skills with approximate minimum cost
  Let U be the set of uncovered skills (initially all skills)
  Let T be the selected team (initially empty)
  Let $total\_cost \leftarrow 0$
  while $U \neq \emptyset$ **do**
    for each freelancer f_j not in T **do**
      $new\_cover \leftarrow \{s_i \in U : s_i \in T_j \wedge c_{ji} \geq r_i\}$
      $effectiveness \leftarrow w_j / |new\_cover|$
    end for
    Select freelancer f_k with minimum effectiveness ratio
    $T \leftarrow T \cup \{f_k\}$
    $total\_cost \leftarrow total\_cost + w_k$
    $U \leftarrow U \setminus new\_cover_k$
  end while
  return $total\_cost, T$

2.2.2 Approximation Ratio Analysis

Theorem 6 (Greedy Approximation Bound)**.** *The greedy algorithm achieves an approximation ratio of H_d, where $d = \max_j |T_j|$ is the maximum number of skills covered by any freelancer, and $H_d = \sum_{i=1}^{d} \frac{1}{i}$ is the d-th harmonic number.*

Proof. Let OPT be the optimal cost and $GREEDY$ be the cost of the greedy solution.

We use the charging argument: when the greedy algorithm selects a freelancer covering k new skills at cost c, we charge $\frac{c}{k}$ to each newly covered skill.

Let $C(s_i)$ be the total charge to skill s_i over the algorithm's execution. Then:

$$GREEDY = \sum_{i=1}^{n} C(s_i)$$

Now, consider the optimal solution F^*. For each freelancer $f_j \in F^*$ with cost w_j covering skills T_j, we analyze the charges to skills in T_j.

At any point in the greedy algorithm where skills in T_j are not yet covered, the effectiveness ratio of f_j is $\frac{w_j}{|T_j \cap U|}$, where U is the set of uncovered skills.

By the greedy choice property, when the algorithm selects a freelancer covering k skills at cost-per-skill α, we have:

$$\alpha \leq \frac{w_j}{|T_j \cap U|}$$

for all available freelancers f_j.

Now, order the skills in T_j by when they were covered in the greedy algorithm: $s_1, s_2, \ldots, s_{|T_j|}$. When skill s_k is covered, at least $|T_j| - k + 1$ skills from T_j remain uncovered. Therefore, the cost charged to s_k is at most:

$$C(s_k) \leq \frac{w_j}{|T_j| - k + 1}$$

Summing over all skills in T_j:

$$\sum_{s_i \in T_j} C(s_i) \leq w_j \cdot \sum_{k=1}^{|T_j|} \frac{1}{|T_j| - k + 1} = w_j \cdot H_{|T_j|}$$

Since each skill is covered by at least one freelancer in F^*, we have:

$$GREEDY = \sum_{i=1}^{n} C(s_i) \leq \sum_{f_j \in F^*} \sum_{s_i \in T_j} C(s_i) \leq \sum_{f_j \in F^*} w_j \cdot H_{|T_j|} \leq H_d \cdot OPT$$

Thus, $GREEDY \leq H_d \cdot OPT$. □

2.2.3 Tightness of the Bound

Theorem 7 (Tightness)**.** *The H_d approximation bound is tight for the greedy algorithm.*

Proof. This follows from the known tight examples for set cover. Consider an instance with d skills and $d+1$ freelancers:

- Freelancer f_0: covers all skills, cost $= 1 + \epsilon$
- Freelancers $f_1, \ldots, f_d$: each covers one distinct skill, cost $= \frac{1}{i}$ for f_i

The greedy algorithm will select $f_d, f_{d-1}, \ldots, f_1$ in sequence, with total cost H_d, while the optimal solution is f_0 with cost $1 + \epsilon$. As $\epsilon \to 0$, the ratio approaches H_d. □

2.2.4 Time Complexity

Theorem 8 (Greedy Algorithm Complexity)**.** *The greedy algorithm runs in $O(m^2 \cdot n)$ time.*

Proof. In the worst case:

- The while loop runs at most n iterations (one freelancer per skill)
- Each iteration checks all m freelancers
- Each freelancer check requires $O(n)$ operations to compute newly covered skills

Thus total time: $O(n \cdot m \cdot n) = O(m \cdot n^2)$.

However, in practice, the number of iterations is bounded by the number of freelancers selected, which is at most m, giving $O(m^2 \cdot n)$. □

2.2.5 Hybrid Approach

In practice, we can combine the greedy approach with exact methods:

Algorithm 2 Hybrid Greedy-DP Algorithm

```
Use greedy algorithm to get upper bound UB
Run DP algorithm but prune branches exceeding UB
Return best solution found
```

This approach leverages the greedy solution's quality to dramatically reduce the DP search space while maintaining optimality guarantees.

Alternatively, we could use the exact algorithm for small n (most cases) and the greedy for the rest. The correct approach depends of the problem/use-case.

Bibliography

[1] Kleinberg, J., and Tardos, E. Algorithm design. 638–643.

[2] Kleinberg, J., and Tardos, E. Algorithm design. 482–485.