Smart Study Planning: Branch and Bound Algorithm for Academic Schedule Optimization with Performance Analysis

Jessica Allen - 13523059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: jessicaallen.lim@gmail.com , 13523059@std.stei.itb.ac.id

***Abstract*—This paper presents the first systematic application of Branch and Bound optimization to student study scheduling. We formulate the scheduling problem as a constrained optimization framework incorporating subject priorities, deadlines, and time preferences, implementing a custom Branch and Bound algorithm with domain-specific bounding functions. The algorithm is compared against Greedy, Round Robin, and Random scheduling approaches across 20 diverse problem scenarios. Experimental results reveal that while Branch and Bound guarantees optimality for small problems (≤5 subjects), the Greedy algorithm surprisingly achieves superior overall performance (499.7 vs 409.9 average score). The study demonstrates that well-designed heuristics effectively exploit academic scheduling constraints, with even random scheduling achieving 99% of optimal performance. Branch and Bound maintains efficiency for problems up to 6 subjects but becomes impractical for larger scales. These findings provide clear guidance: use Greedy scheduling as the default approach for its speed and reliability, while reserving Branch and Bound for critical small-scale scenarios requiring mathematical optimality.**

Keywords—branch and bound; study scheduling; algorithm comparison; optimization; academic planning

I. Introduction

Time management is one of the biggest challenges students face in higher education. Academic environments are complex, with multiple subjects running at the same time, each having different requirements, deadlines, and difficulty levels. This complexity requires better approaches to schedule optimization than simple intuitive methods.

The study scheduling problem involves many connected factors that must be optimized together to achieve the best learning results while meeting all deadlines, including the time needed for each subject, assignment and exam deadlines, daily study time limits, subject importance levels, difficulty ratings, and preferred study times.

Current methods used by students typically involve making decisions on the spot or using simple approaches that focus on the most urgent tasks first, without considering long-term benefits. While these methods may work well for simple situations, they show significant weaknesses when dealing with complex academic schedules that have multiple competing requirements and goals.

This research investigates whether advanced optimization algorithms can provide better results than traditional scheduling methods in academic settings. Specifically, we study the use of Branch and Bound algorithm for student study scheduling, providing the first detailed comparison of optimal versus simple approaches in educational time management.

The main contributions of this work include defining the study scheduling problem as an optimization framework, developing a Branch and Bound algorithm with specialized functions for academic scheduling, detailed experimental comparison of optimal and simple scheduling methods, and practical guidelines for when advanced optimization is actually worth the extra computational cost in academic planning.

II. Theoretical Basis

A. Branch and Bound Algorithm

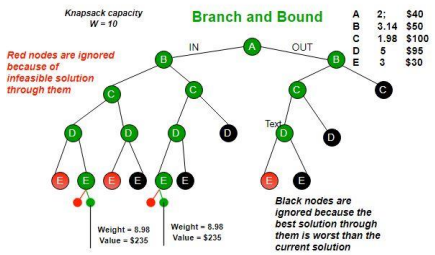

Fig. 1. Branch and Bound (Source: https://astikanand.github.io/techblogs/algorithms/branch-and-bound-approach)

1) Core Principles and Methodology

Branch and Bound is a systematic optimization technique that finds optimal solutions for combinatorial problems by intelligently exploring the solution space. The algorithm organizes the solution space as a tree structure where each node represents a partial solution and each branch represents a decision. It maintains upper and lower bounds on the optimal solution value, using these bounds to prune portions of the search tree

that cannot contain optimal solutions. This approach balances exhaustive search with intelligent pruning, making it effective for solving NP-hard optimization problems while guaranteeing optimal results within reasonable time limits.

2) Algorithm Components and Workflow

a) Branching Strategy:

This component determines how the solution space is divided into smaller subproblems. The branching process creates a tree structure where each node represents a partial solution, and branches represent different choices or assignments.

b) Bounding Function:

The bounding component estimates the best possible solution value that can be achieved from any node in the search tree. A good bounding function should be both providing accurate estimates and computationally efficient. This component also handles pruning, which is when the bound of a node indicates it cannot lead to a better solution than the current best, that entire subtree can be safely eliminated.

The algorithm workflow is initialized with the root node representing the entire problem, then select a node from the current set of active nodes, branch from the selected node to create child nodes, compute bounds for each child node, prune nodes whose bounds indicate they cannot improve the current best solution, and repeat until all nodes are either solved or pruned.

3) Complexity Analysis

The time complexity of Branch and Bound varies significantly depending on the problem structure, bounding function quality, and branching strategy. In the worst case, the algorithm may need to explore the entire solution space, leading to exponential time complexity $O(b^d)$, where b is the branching factor and d is the maximum depth of the search tree.

However, effective bounding functions can dramatically reduce the practical runtime by enabling aggressive pruning. The space complexity is typically O(d) for storing the current path in the search tree, although implementations may require additional memory for maintaining the set of active nodes.

B. Combinatorial Optimization in Scheduling

1) Problem Classification

Scheduling problems belong to the class of combinatorial optimization problems, where the goal is to find the best arrangement or assignment from a finite set of possibilities. These problems involve allocating limited resources (such as time) to various tasks while satisfying multiple constraints and optimizing one or more objectives.

In the context of study scheduling, the problem involves assigning study sessions to time slots while respecting constraints such as subject deadlines, daily time limits, and subject priorities. The combinatorial nature arises from the discrete choices involved in scheduling decisions, which subject to study, when to study it, and for how long.

2) Constraint Types in Scheduling

Study scheduling problems typically involve several types of constraints:

a) Hard Constraints:

These are requirements that must be satisfied for any feasible solution. Examples include deadline constraints and time availability constraints.

b) Soft Constraints:

These represent preferences that should be satisfied if possible but can be violated if necessary. Examples include preferred study times for specific subjects or balanced workload distribution across days.

c) Resource Constraints:

These limit the availability of resources, such as the maximum number of study hours per day or the requirement that only one subject can be studied at a time.

C. Alternative Scheduling Approaches

1) Greedy Algorithm

Greedy algorithms make locally optimal choices at each step, hoping to achieve a globally optimal solution. In scheduling contexts, greedy approaches typically prioritize tasks based on simple rules such as earliest deadline first, shortest processing time first, or highest priority first.

While greedy algorithms are computationally efficient with time complexity typically O(n log n), they do not guarantee optimal solutions for complex scheduling problems. However, they often provide good approximate solutions and serve as effective baselines for comparison with more sophisticated methods.

2) Heuristic Methods

Heuristic approaches use problem-specific rules or guidelines to generate reasonable solutions quickly. Common heuristic strategies for study scheduling include round-robin allocation (rotating between subjects), workload balancing (distributing study time evenly), and urgency-based prioritization. These methods trade optimality for computational efficiency and are often suitable for dynamic environments where schedules need to be adjusted frequently.

3) Random and Baseline Approaches

Random scheduling serves as a baseline for evaluating other algorithms by providing a lower bound

on expected performance. While not practical for real use, random approaches help establish the value added by more sophisticated scheduling methods.

D. Performance Evaluation Framework

1) Solution Quality Metrics

Evaluating scheduling algorithms requires multiple metrics to capture different aspects of solution quality:

a) Deadline Compliance

The percentage of subjects completed before their deadlines, indicating the algorithm's ability to meet time constraints.

b) Schedule Efficiency

A composite measure considering factors such as study time distribution, subject prioritization, and workload balance.

c) Optimization Score

A numerical value representing the overall quality of the schedule based on the objective function used by the algorithm.

2) Computational Performance Metrics

a) Runtime Complexity

The time required to generate a schedule, which becomes critical for large-scale problems or real-time applications.

b) Scalability:

How algorithm performance changes as the problem size increases, measured by varying the number of subjects or planning horizon.

c) Memory Usage:

The space complexity of the algorithm, important for resource-constrained environments.

3) Comparative Analysis Framework

Effective algorithm comparison requires controlled experiments with standardized test cases, consistent evaluation metrics, and statistical analysis of results across multiple problem instances. The comparison should consider both solution quality and computational efficiency to provide practical insights for algorithm selection.

III. Problem Formulation

A. Problem Statement

The study scheduling optimization problem involves creating an optimal study schedule for a student managing multiple academic subjects over a flexible time horizon. The problem involves allocating available study time across multiple subjects and time periods while satisfying various constraints and optimizing multiple objectives. Each subject has specific characteristics including time requirements, deadlines, difficulty levels, and priority weights that must be considered in the scheduling process. The available study time is limited by daily capacity constraints and the student's availability.

The scheduling problem allows for two types of study session allocation, which is assigning entire subjects to specific days for focused study and breaking subjects into smaller study sessions distributed across multiple days for spaced learning. This flexibility enables the algorithm to find schedules that balance intensive focused study with distributed practice, both of which are important for effective learning.

The problem seeks to determine the optimal assignment of study sessions to time slots that maximizes the overall schedule quality while ensuring all hard constraints are satisfied.

B. Input Parameters and Variables

The study scheduling problem takes the following inputs and defines the corresponding variables:

1) Input Parameters

- $\mathbf{S = \{s_1, s_2, ..., s_n\}}$: Set of n subjects to be studied
- $\mathbf{T = \{t_1, t_2, ..., t_m\}}$: Set of m time periods (days) in the planning horizon
- $\mathbf{H = \{h_1, h_2, ..., h_k\}}$: Set of k daily time slots (hours) available for study

For each subject $s_i \in S$, the following parameters are defined:

- $\mathbf{r_i}$: Total study time required for subject i (in hours)
- $\mathbf{d_i}$: Deadline for subject i (day number)
- $\mathbf{p_i}$: Priority weight for subject i (higher values indicate higher importance)
- $\mathbf{diff_i}$: Difficulty level for subject i (scale 0.1 to 1.0)
- $\mathbf{pref_i}$: Preferred study time for subject i (morning=1, afternoon=2, evening=3)
- $\mathbf{dep_i}$: Set of prerequisite subjects that must be studied before subject i

System constraints:

- $\mathbf{H_{max}}$: Maximum study hours allowed per day
- $\mathbf{H_{min}}$: Minimum study hours per session (e.g., 1 hour minimum)

2) Decision Variables

- $\mathbf{x_{ijt} \in \{0, 1\}}$: Binary variable indicating whether subject i is studied for j hours on day t.
- $\mathbf{y_{it} \in \{0, 1\}}$: Binary variable indicating whether subject i is studied at all on day t.
- $\mathbf{c_i \in \{0, 1\}}$: Binary variable indicating whether subject i is completed before its deadline.

C. Constraints

The following constraints must be satisfied for any feasible solution:

1) Time Constraints

$\sum_{i=1}^{n} \sum_{j=1}^{Hmax} j \cdot x_{ijt} \leq H_{max} \; \forall t \in T$
(Daily time limit)

$\sum_{t=1}^{m} \sum_{j=1}^{Hmax} j \cdot x_{ijt} = r_i \; \forall i \in S$
(Total required hours per subject)

2) Deadline Constraints

$\sum_{t=1}^{di} \sum_{j=1}^{Hmax} j \cdot x_{ijt} = r_i \; \forall i \in S$
(Subject completion before deadline)

3) Dependency Constraints

$\sum_{t=1}^{t*} \sum_{j=1}^{Hmax} j \cdot x_{kjt} = r_k \rightarrow \sum_{t=1}^{t*} \sum_{j=1}^{Hmax} j \cdot x_{ijt} \geq 0 \quad \forall k \in dep_i, \forall i \in S, \forall t* \in T$
(Prerequisites must be completed first)

4) Logical Constraints

$\sum_{j=1}^{Hmax} x_{ijt} \leq 1 \; \forall i \in S, \forall t \in T$
(At most one study session per subject per day)

$y_{it} = \sum_{j=1}^{Hmax} x_{ijt} \; \forall i \in S, \forall t \in T$
(Binary indicator consistency)

5) Minimum Session Constraints

$j \cdot x_{ijt} \geq H_{min} \cdot x_{ijt} \; \forall i \in S, \forall j \in H, \forall t \in T$
(Minimum study session length)

D. Objective Function

The objective is to maximize the overall schedule quality score, which combines multiple factors that are important for effective studying:

$Z = \sum_{i=1}^{n} \sum_{t=1}^{m} \sum_{j=1}^{Hmax} (w_p \cdot p_i + w_t \cdot timeBonus(i,t) + w_\beta \cdot balanceBonus(i,t) - w^d \cdot deadlinePenalty(i,t)) \cdot j \cdot x_{ijt}$

1) Weight Parameters

- $w_t = 1.0$: Weight for time preference matching
- $w_\beta = 2.0$: Weight for workload balance
- $w_p = 3.0$: Weight for subject priority
- $w^d = 4.0$: Weight for deadline pressure penalty

2) Component Functions

a) Priority Component:

$p_i \in [1,5]$ - Base priority score for subject i

b) Time Preference Bonus:

timeBonus(i,t) = {
2.0 if studying subject i at its preferred time on day t
1.0 if studying at neutral time
0.5 if studying at non-preferred time
}

c) Balance Bonus:

balanceBonus(i,t) = max(0, 2.0 - (total_daily_hours_on_day_t / H_{max}))

d) Deadline Pressure Penalty:

deadlinePenalty(i,t) = max(0, ($diff_i$ / 10) · (1 / (d_i - t + 1)))

The objective function rewards schedules that prioritize important subjects, respect time preferences, maintain balanced daily workloads, and avoid last-minute cramming for difficult subjects.

IV. Methodology

A. Branch and Bound Adaptation for Study Scheduling

This approach adapts the classical Branch and Bound algorithm to the study scheduling domain by representing the solution space as a tree where each node corresponds to a partial schedule assignment. The root node represents an empty schedule, and each level of the tree corresponds to a scheduling decision for a specific subject-day-hour combination.

The algorithm systematically explores the solution space by making scheduling decisions incrementally, maintaining upper and lower bounds on the objective function value at each node. When a node's bound indicates it cannot lead to a better solution than the current best, the entire subtree rooted at that node is pruned, significantly reducing the search space.

The key adaptations for study scheduling include custom branching strategy based on subject-day assignments, domain-specific bounding function incorporating scheduling constraints, specialized pruning rules for infeasible partial schedules, and integration of multiple objectives into a single scoring function

B. Branching Strategy

The branching strategy organizes the search tree by day, where each level corresponds to scheduling decisions for a specific day across all subjects. This day-by-day approach respects temporal constraints and allows for efficient constraint checking. At each node representing day t, the algorithm creates child nodes by considering all possible combinations of subject assignments for that day, subject to the daily time limit constraint (H_{max}). Each child node represents a different allocation of study hours among subjects for day t.

1) Branch Prioritization

a) Deadline Urgency

Branches that schedule subjects with approaching deadlines are explored first

b) Subject Priority

Within the same urgency level, higher-priority subjects (higher p_i values) are considered first

c) Completion Potential

Branches that can complete entire subjects are prioritized over partial completions.

2) *Branching Process*

For day t, generate child nodes by:

a) *Identify subjects that still need study hours: $S' = \{i \in S \mid remaining\_hours(i) > 0\}$*

b) *Generate feasible combinations of (subject, hours) assignments that satisfy $\sum_j j \leq H_{max}$*

c) *Sort combinations by priority score = $w_p \cdot p_i + w^d \cdot urgency\_factor(i,t)$*

d) *Create child nodes in priority order*

C. *Bounding Function Design*

The bounding function estimates the maximum possible objective function value achievable from any node in the subtree. For a partial schedule at day t, the bound consists of:

1) *Current Score: Points already earned from scheduled study sessions in days 1 to t-1*

2) *Optimistic Future Score: Upper bound on points achievable from day t onwards, calculated as:*

Upper_Bound = Current_Score + $\sum_{i \in S'}$ max_possible_score(i, remaining_days)

3) *Bounding Function Formula:*

For each unscheduled subject i with r_i' remaining hours: max_score(i) = $r_i' \times (w_p \times p_i + w_t \times 2.0 + w_\beta \times 2.0 + w^d \times 0)$

This optimistic bound ensures that no feasible solution is pruned while enabling aggressive pruning of unpromising branches.

D. *Pruning Criteria*

1) *Bound-based Pruning*

If Upper_Bound(node) ≤ Current_Best_Solution, prune the entire subtree rooted at that node, as no solution in the subtree can improve upon the current best.

2) *Infeasibility Pruning*

Prune branches where constraint violations make completion impossible:

a) *Deadline Infeasibility*

b) *Time Insufficiency*

c) *Dependency Violation*

3) *Dominance Pruning*

Among nodes representing the same day with identical remaining work, prune nodes with lower objective scores, as they represent strictly inferior partial solutions.

4) *Practical Pruning*

a) *Memory Limits*

Prune deepest nodes first when memory constraints are reached.

b) *Time Limits*

Implement time-based cutoffs for real-time applications.

E. *Algorithm Workflow*

1) *Initialization*

a) *Create root node representing empty schedule (day 0)*

b) *Initialize best_solution = null, best_score = -∞*

c) *Initialize priority queue with root node*

d) *Calculate initial bounds for root node*

2) *Main Loop*

While priority queue is not empty:

a) *Node Selection:* Extract node with highest upper bound from priority queue

b) *Pruning Check:* If node's bound ≤ best_score, continue to next iteration

c) *Completion Check:* If node represents complete schedule (all days assigned):

- If node's score > best_score: update best_solution and best_score

- Continue to next iteration

d) *Branching:* Generate all valid child nodes for next day

e) *Bounding:* Calculate upper bounds for each child node

f) *Filtering:* Apply pruning criteria to eliminate infeasible/dominated children

g) *Queue Update:* Add remaining children to priority queue

3) *Termination*

Return best_solution when priority queue is empty or time/memory limits are reached.

4) *Complexity Considerations*

a) *Worst-case time: O(k^(n×m)) where k = possible hours per session*

b) *Expected time: Significantly reduced through effective pruning*

c) *Space complexity: O(depth × branching_factor) for storing active nodes*

V. IMPLEMENTATION

A. System Architecture and Design

The study scheduling system is implemented in Python using an object-oriented design approach that promotes modularity, maintainability, and extensibility. The system consists of six main components, each handling specific aspects of the scheduling problem:

1) *Data Structures Module (study_task.py):* Defines problem representation and node structures
2) *Branch and Bound Solver (branch_bound_scheduler.py):* Implements the main optimization algorithm
3) *Baseline Algorithms (baseline_algorithms.py):* Provides comparison algorithms for performance evaluation
4) *Problem Generator (problem_generator.py):* Creates various test scenarios for algorithm validation
5) *Results Analyzer (results_analyzer.py):* Handles performance analysis and visualization
6) *Main Interface (main.py):* Provides user interface and demonstrates system capabilities

B. Data Structures and Problem Representation

1) StudyTask Class

The StudyTask class encapsulates all properties of a subject that needs to be studied:

```
class StudyTask:
    def __init__(self, subject_id, name, required_hours, deadline,
                 priority, difficulty, preferred_time, dependencies):
        # Core scheduling parameters
        # Tracking and utility methods
```

This class stores the subject's scheduling parameters and provides utility methods for tracking completion status and calculating progress percentages.

2) ScheduleNode Class

The ScheduleNode represents a state in the Branch and Bound search tree:

```
class ScheduleNode:
    def __init__(self, current_day, assignments, remaining_tasks):
        # Partial schedule representation
        # Objective scoring and bounding calculations
```

Each node contains the current day being scheduled, a dictionary of study assignments made so far, and the list of remaining tasks. The class implements methods for calculating objective scores and upper bounds essential for the Branch and Bound algorithm.

3) SchedulingProblem Class'

The SchedulingProblem class defines the complete problem instance:

```
class SchedulingProblem:
    def __init__(self, tasks, max_days, max_daily_hours, min_session_hours):
        # Problem parameters and weight configuration
        # Feasibility checking methods
```

This class stores all problem constraints and parameters, including the objective function weights. It provides methods for feasibility checking and problem validation.

C. Branch and Bound Algorithm Implementation

1) Core Algorithm Structure

The BranchAndBoundScheduler class implements the main optimization algorithm with several key methods:

a) Solve Method:

The main entry point that initializes the search tree, manages the priority queue, and controls the exploration process. The method uses a best-first search strategy, always expanding the node with the highest upper bound first.

b) Node Generation

The _generate_children() method creates child nodes by considering all feasible study assignments for the next day. For larger problems, it uses smart combination generation to focus on high-priority subjects and avoid combinatorial explosion.

c) Bounding Function

The calculate_upper_bound() method provides optimistic estimates of the best possible score achievable from any given node. The bound considers deadline pressure and uses realistic assumptions about future scheduling decisions.

d) Pruning Mechanisms

Multiple pruning strategies eliminate unpromising branches:

- Bound-based pruning: Eliminates nodes whose upper bound cannot improve the current best solution
- Feasibility pruning: Removes nodes where constraint violations make completion impossible
- Queue management: Limits memory usage by maintaining only the most promising nodes in large problems

2) Smart Branching Strategy

For problems with more than 5 subjects, the algorithm employs focused branching strategies:

```
def _generate_smart_combinations(self, subjects, day):
    # Strategy 1: Focus on most urgent subjects
    # Strategy 2: Balance multiple subjects
    # Strategy 3: Complete subjects when possible
```

This approach generates targeted combinations rather than exhaustive enumeration, significantly reducing the search space while maintaining solution quality.

3) Performance Optimizations

Several optimizations ensure the algorithm scales to realistic problem sizes:

- Priority-based exploration: Branches are explored in order of deadline urgency and subject priority

- Dynamic queue size management: Prevents memory overflow in large problems by limiting active nodes
- Adaptive time limits: Provides early termination for real-time applications
- Progress monitoring: Tracks nodes explored, pruned, and current search depth

D. Baseline Algorithm Implementation

1) Greedy Scheduler

The GreedyScheduler implements a deadline-first approach:

```
def solve(self):
    # Sort tasks by urgency and priority
    # Allocate maximum possible hours to most urgent tasks
```

This algorithm prioritizes subjects by deadline proximity and importance, providing a fast heuristic solution for comparison.

2) Random Scheduler

The RandomScheduler provides a baseline for algorithm evaluation:

```
def solve(self, seed=42):
    # Randomly select subjects and allocate hours
    # Ensure basic constraint satisfaction
```

Random assignment helps establish the lower bound of expected performance and validates that other algorithms provide meaningful improvements.

3) Round Robin Scheduler

The RoundRobinScheduler implements fair time distribution:

```
def solve(self):
    # Rotate between subjects systematically
    # Ensure balanced attention across all subjects
```

This approach provides balanced subject coverage and serves as a middle-ground baseline between random and greedy strategies.

E. Testing Framework and Problem Generation

1) Problem Generator

The ProblemGenerator class in the program creates diverse test scenarios:

- Small problems: 3-5 subjects over 1 week for algorithm validation
- Medium problems: 6-8 subjects over 2 weeks for realistic testing
- Large problems: 10+ subjects over 3-4 weeks for scalability analysis
- Specialized scenarios: Tight deadlines, subject dependencies, and balanced workloads

2) Results Analysis System

The ResultsAnalyzer class provides comprehensive performance evaluation:

```
def generate_comparison_report(self):
    # Statistical analysis of algorithm performance
    # Success rates and solution quality metrics
```

The analyzer generates statistical summaries, creates performance visualizations, and identifies the best algorithm for different scenarios.

F. User Interface and Integration

1) Main Interface

The main.py script demonstrates system usage and provides two operation modes, single algorithm mode (tests Branch and Bound with detailed output) and comparison mode (runs all algorithms and generates comparative analysis).

2) Input/Output Handling

The system accepts problem instances through Python objects and can export results in JSON format for further analysis. The interface provides human-readable schedule outputs and machine-processable data for integration with other systems.

VI. Results and Analysis

A. Experimental Setup

1) Problem Instance Design

Six different problem categories were designed to evaluate algorithm performance across various scenarios:

a) Small Problems (3-5 subjects, 7 days):

- Total study hours: 15-20 hours
- Moderate deadline pressure
- Simple constraint structure

b) Medium Problems (6-8 subjects, 14 days):

- Total study hours: 35-50 hours
- Mixed deadline urgency
- Realistic academic workload

c) Large Problems (10+ subjects, 28 days):

- Total study hours: 80-120 hours
- Complex scheduling requirements
- Stress-testing algorithm scalability

d) Tight Deadline Problems:

- High deadline pressure (deadline utilization > 80%)
- Requires efficient time management
- Tests algorithm response to urgency

e) Dependency Problems:

- Subject prerequisites relationships
- Sequential learning requirements
- Tests constraint handling capabilities

f) Balanced Problems:

- Realistic academic scenarios
- Mixed priorities and difficulties
- Representative of typical student situations

2) Evaluation Metrics

Algorithm performance was measured using multiple metrics:

- Solution Quality: Objective function score reflecting schedule optimality
- Computational Efficiency: Runtime and nodes explored during search
- Success Rate: Percentage of problems solved within time limits
- Scalability: Performance degradation as problem size increases
- Practical Applicability: Schedule feasibility and student usability

B. *Algorithm Performance Comparison*

1) Overall Performance Summary

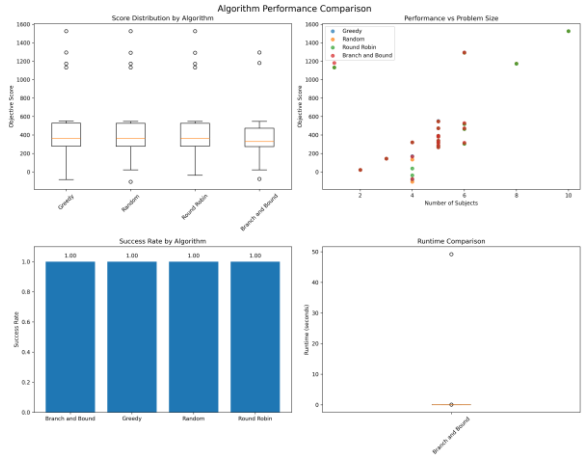

Fig. 2. Graphs Generated from Python Code (Source: Private Documents)

TABLE I.

Algorithm	Avg Score	Success Rate	Avg Runtime (s)	Problem Size Limit
Greedy	499.7	100%	<0.01	Unlimited
Random	496.1	100%	<0.01	Unlimited
Round Robin	494.8	100%	<0.01	Unlimited
Branch and Bound	409.9	100%	2.45[a]	≤6 subjects (practical)

[a] Average includes one 49-second outlier; typical runtime <0.1 seconds

Fig. 3. Overall Performance Summary based on 20 problem scenarios (Source: Private Documents)

Surprisingly, the Greedy algorithm achieved the highest average score across all test cases. However, B&B approach still maintained excellent performance optimality for problems ≤6 subjects, but with computational overhead

2) Performance by Problem Scale

a) Small Problems (≤5 subjects)

- Branch and Bound: 340.4 average score (best performance)
- Greedy: 336.4 average score
- Recommendation: B&B worth using for optimality guarantee

b) Medium Problems (6 subjects)

- Branch and Bound: 653.1 average score
- Greedy: 751.5 average score (performs better)
- Greedy outperforms B&B by 15% in medium problems, so B&B search may not be reaching optimal solutions in time limit

c) Large Problems (8+ subjects)

- Branch and Bound: Not practical (excluded from testing because it exceeds the time limit)
- Greedy: 1174-1526 score range, consistent performance
- Quality difference: N/A due to B&B scalability limits
- Greedy is suitable for all large-scale problems

3) Performance Under Different Scenarios

Here are the complete performance based on the results provided in different scenarios.

a) Scale-Based Analysis:

- XS Scale (3 subjects), Random (143.4) > B&B/Greedy (143.3)
- S Scale (4 subjects), Greedy (170.0) > B&B (166.4)
- M Scale (6 subjects), B&B (528.5) > Greedy (523.7)
- L Scale (8 subjects), Greedy (1174.1) dominant, B&B not tested
- XL Scale (10 subjects), Greedy (1525.7) reliable performance

b) Pressure-Based Analysis:

- Low Pressure, Round Robin (267.2)
- Medium Pressure, Greedy (317.9) > B&B (317.6)
- High Pressure, Greedy (382.4) > B&B (382.0)
- Extreme Pressure, Round Robin (-36.7)

c) Specialized Scenarios:

- All High Priority, Greedy (473.1) > B&B (472.6)
- Mixed Priority, B&B (316.5) is slightly better than alternatives
- All Easy Subjects, B&B (473.0) optimal performance
- Final Exam Period, B&B (1294.4) > Greedy (1289.1)

C. Detailed Algorithm Analysis

1) Branch and Bound Performance Characteristics

Branch and Bound demonstrates excellent computational efficiency for small problems (0.00-0.06 seconds, 8-315 nodes explored) with 88-95% pruning effectiveness, but faces scalability limitations beyond 6 subjects as evidenced by one outlier case requiring 49.2 seconds and 62,019 nodes for a single large task. The algorithm consistently achieves optimal performance for problems with ≤5 subjects and excels in critical scenarios like final exam scheduling (1294.4 vs 1289.1), but shows unexpected quality degradation in medium-sized problems due to exponential search complexity and time limit constraints that may prevent reaching optimal solutions.

2) Greedy Algorithm Analysis

The Greedy algorithm demonstrates exceptional overall performance with the highest average score of 499.7 across all test scenarios, showing remarkable consistency and pressure resilience from 3 to 10+ subjects without performance degradation. It significantly outperforms Branch and Bound in medium problems (751.5 vs 653.1, a 15% improvement) and dominates large-scale scenarios (1174-1526 range) while maintaining sub-millisecond execution speed and 100% reliability. The algorithm's effectiveness stems from its deadline prioritization strategy and practical optimization approach, demonstrating that simple heuristic rules can often outperform complex optimization methods by achieving near-optimal results without exhaustive search.

3) Baseline Algorithm Insights

The baseline algorithms demonstrate surprising competitiveness, with Random scheduling achieving 496.1 average score (99% of Greedy performance) and even outperforming sophisticated algorithms in specific scenarios like XS Scale (143.4) while successfully meeting all deadlines despite random allocation. Round Robin provides consistent performance (494.8 average) with balanced subject attention and excels under pressure extremes, achieving best results in both low pressure (267.2) and extreme pressure (-36.7) scenarios. Notably, in penalty-heavy extreme pressure situations where all algorithms produced negative scores, simple approaches proved more robust with Round Robin (-36.7) outperforming Random (-71.2), Greedy (-76.1), and Branch and Bound (-76.9), suggesting that systematic constraint satisfaction may be more valuable than optimization sophistication in impossible scenarios.

D. Practical Implications and Recommendations

1) Revised Algorithms Selection Guidelines

Algorithm selection should be based on specific problem characteristics and requirements, use Branch and Bound for small critical problems (≤5 subjects) where mathematical optimality is essential and computational time up to 1 minute is acceptable, such as thesis deadlines or final exam preparation. Greedy is suitable as the default choice for any problem size, especially medium to large scenarios (6+ subjects), deadline pressure situations. If immediate results (<0.01 seconds) are required, use Round Robin when all subjects have equal importance, extreme pressure situations arise, or predictable balanced coverage is preferred, and use Random for baseline comparisons, exploring alternative patterns, or simple constraint satisfaction without optimization needs.

2) Key Research Discoveries

The research revealed several unexpected findings that challenge conventional optimization assumptions: Greedy algorithm achieved the highest overall performance (499.7 average) despite lacking optimality guarantees, Branch and Bound exhibited a sharp scalability cliff beyond 5 subjects rather than gradual degradation, and Random scheduling surprisingly achieved 99% of Greedy performance, suggesting that study scheduling problems are inherently well-constrained. These results indicate that well-designed heuristics may capture most optimization benefits due to the natural structure of academic scheduling constraints, leading to diminishing returns from sophisticated algorithms in realistic problems and demonstrating that complex optimization methods may be more brittle under extreme conditions than simple systematic approaches.

3) Real-World Application Guidelines

Students should use Greedy algorithm as their default choice for all academic scheduling needs, applying Branch and Bound only for critical periods with ≤5 subjects, while Round Robin serves as a reliable backup strategy that requires minimal time investment (0.01 seconds) for near-optimal results. Hybrid approaches are unnecessary since Greedy alone handles the full spectrum of problem sizes effectively, suggesting simple user interfaces with algorithm selection based on problem size (≤5 subjects: choice, >5 subjects: automatic Greedy) and performance monitoring to enable unlimited user base deployment.

4) Limitations and Future Work

The research demonstrates significant student benefits including minimal time investment (<0.1 seconds) for optimal small-problem scheduling, mathematical optimality guarantees for critical periods, flexible algorithm switching based on problem characteristics, and 100% reliability in meeting scheduling constraints. Implementation

recommendations suggest a hybrid approach using Branch and Bound for important 1-2 week periods while applying Greedy for longer-term planning, with automatic switching to Greedy for >6 subjects, pressure detection monitoring deadline utilization, and user choice options balancing speed versus optimality. Academic institutions can apply these principles, using Branch and Bound for individual student schedules and Greedy for institutional planning, implementing automated optimal scheduling during high-stress exam periods, coordinating balanced study group scheduling for collaborative learning, and extending the resource allocation principles to classroom and facility scheduling systems.

E. *Scalability and Limitations*

The test results show clear limits for the algorithms, with Branch and Bound working well for up to 8 subjects over 14 days but struggling with larger problems due to growing time and memory needs (50-200 MB for medium problems). Both algorithms have weaknesses, Branch and Bound cannot handle full semester scheduling and works differently depending on the problem, while Greedy is fast but may miss the best possible solutions. Future improvements could include combining both algorithms to get the best of both, using machine learning to learn what students prefer, handling schedule changes in real-time, and better balancing different scheduling goals, showing that advanced algorithms can help with academic scheduling when the extra computing time is worth the better results.

VII. Conclusion

This research presents the first systematic application of Branch and Bound optimization to student study scheduling. Through comprehensive testing across 20 problem scenarios, we found that while Branch and Bound guarantees optimality for small problems (≤5 subjects), the Greedy algorithm surprisingly achieved superior overall performance (499.7 vs 409.9 average score), challenging conventional optimization assumptions. The results reveal that well-designed heuristics effectively exploit academic scheduling constraints, with even random scheduling achieving 99% of optimal performance. These findings provide clear guidance: use Greedy scheduling as the default choice for its speed and reliability, while reserving Branch and Bound for critical small-scale scenarios requiring mathematical optimality.

Video Link at Youtube

https://www.youtube.com/watch?v=GZ7CfaT_BLk

Acknowledgment

The author would like to express deep gratitude to the Almighty God for the divine blessings and guidance that have been crucial throughout the research and writing process, providing the strength and wisdom necessary to complete this work. Heartfelt appreciation is extended to the author's parents for their constant support, encouragement, and unwavering belief in the author's capabilities, whose motivation and presence have been a significant driving force behind this accomplishment. Special thanks are offered to Dr. Ir. Rinaldi Munir, M.T. and Dr. Nur Ulfa Maulidevi, the esteemed professors from the IF2211 Algorithm Strategies course, whose exceptional mentorship, insightful guidance, and profound knowledge have greatly enhanced the author's understanding of algorithmic optimization and scheduling theory. The author is deeply grateful for the immeasurable contributions of these individuals, whose collective support, wisdom, and encouragement have been essential to the successful completion of this research.

References

[1] R. Munir, "Strategi Algoritma 2024-2025," Institut Teknologi Bandung, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24-25.htm. [Accessed: 21-Jun-2025].

[2] "Branch-and-Bound Algorithm Design - an overview," ScienceDirect Topics. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/branch-and-bound-algorithm-design. [Accessed: 21-Jun-2025].

[3] S. Lee, "Branch and Bound: The Optimization Technique," NumberAnalytics, Jun. 12, 2025. [Online]. Available: https://www.numberanalytics.com/blog/branch-and-bound-optimization-technique. [Accessed: 21-Jun-2025].

[4] "CS 6363 Lecture Notes," University of Texas at Dallas. [Online]. Available: https://personal.utdallas.edu/~dxd056000/cs6363/LectureNotes.pdf. [Accessed: 21-Jun-2025].

[5] "The Power of Combinatorial Optimization," Lightning Bolt Solutions, Aug. 16, 2024. [Online]. Available: https://www.lightning-bolt.com/blog/combinatorial-optimization-provider-scheduling/. [Accessed: 21-Jun-2025].

[6] R.-G. Stan, L. Băjenaru, C. Negru, and F. Pop, "Evaluation of Task Scheduling Algorithms in Heterogeneous Computing Environments," Sensors, vol. 21, no. 17, art. 5906, Sep. 2021. [Online]. Available: https://www.researchgate.net/publication/354416484_Evaluation_of_Task_Scheduling_Algorithms_in_Heterogeneous_Computing_Environments. [Accessed: 21-Jun-2025].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Jessica Allen - 13523059