2019

An Empirical Analysis of an Algorithm for the Budgeted Maximum Vertex Cover Problem in Trees

Mujidat Abisola Adeyemo
West Virginia University, maa0073@mix.wvu.edu

2019

An Empirical Analysis of an Algorithm for the Budgeted Maximum Vertex Cover Problem in Trees

Mujidat Abisola Adeyemo

An Empirical Analysis of an Algorithm for the Budgeted Maximum Vertex Cover Problem in Trees

Mujidat Abisola Adeyemo

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Elaine M. Eschen, Ph.D.
Vahan V. Mkrtchyan, Ph.D
K. Subramani, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2019

Keywords: Budgeted maximum coverage problem, Maximum coverage, Dynamic programming, Trees, Empirical analysis

Abstract

An Empirical Analysis of an Algorithm for the Budgeted Maximum Vertex Cover Problem in Trees

Mujidat Abisola Adeyemo

Covering problems are commonly studied in fields such as mathematics, computer science and engineering. They are also applicable in the real world, e.g., given a city, can we build base-stations such that there is network availability everywhere in the city. However, in the real world, there are usually constraints such as cost and resources. The Budgeted Maximum Vertex Cover is a generalization of covering problems. It models situations with constraints. In this thesis, we empirically analyze an algorithm for the problem of finding the Budgeted Maximum Vertex Cover in undirected trees (BMVCT). The BMVCT problem is defined as follows: Given a tree $T = \langle V, E \rangle$, each vertex is associated with a cost $c : V \to \mathbb{N}$, each edge with a profit given by $p : E \to \mathbb{N}$, and a constraint budget ≥ 0. The goal of BMVCT is to find the subset of vertices with total cost sum $c(u) \leq B$ such that the total profit of the covered edges is maximized. The BMVCT problem occurs in a number of domains such as in transportation or building network base stations and can be adapted to solve the partial vertex cover problem, $\frac{B}{\sum_{u \in V}}$. In particular, we implement the algorithm in [1]. We test the implementation with star trees, random trees and binary trees. This implementation can be adapted to solve the weighted partial vertex cover on trees (WPVCT) as discussed in [1].

Acknowledgements

All praises and adorations go to Almighty Allah by whose grace and favor this thesis was successfully completed. To my thesis advisor, Dr. Subramani, I am deeply grateful to you for your fatherly role, support and motivation in my research and report writing. Your guidance and corrections helped me in completing this research and thesis report. Thank you for showing compassion, empathy and helping me through my weaknesses. It is an honor to have Dr. Eschen and Dr Mkrtchyan as my thesis committee members. Thank you Dr. Eschen, for your support and assistance. You are a mother, friend and role-model to me. Thank you Dr. Mkrtchyan for always making time out for me despite the time difference. Your careful scrutiny and insightful comments helped me in my implementation and in writing this report.

To my mentor and academic advisor, Dr. Adjeroh, I am deeply grateful to you for your encouragement, guidance and advice through my masters studies. To my role-model, Dr. Santiago, thank you for your listening ears and guidance. You are God-sent. To the First year program, FEP Statler College and Ali Anderson, thank you for taking me as family and showing me love. I enjoyed working with you. To my FEP GTA colleagues, thank you for the comradeship. To the Lane Department staff, thank you so much. I thank my professors and all who have impacted me in one way or another. To Chris Randall, thank you for your support.

To my husband and daughter, thank you for your love and emotional support. I could not have done this without you. To my parents, your prayers, love and guidance made me who I am today. I can never repay what you have done for me. To my siblings, I say a big thank you for all you have done. You are the best. To all my friends especially Oba and family, the Animashauns, thank you for everything. I love you all. To the Nigerian community in Morgantown, thank you for your love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Covering problems are computational problems that ask whether a certain combinatorial structure covers another, or how large the structure has to be to do that. Covering problems are abundant in the world we live in and drive many things we do. Our cellphone utilize network coverage. Network companies have to locate their base-stations to enable a maximum distribution of signals based on a budget. We drive through roads or enter buses. Road networks, bus routes, bus stops and bus times are planned to enable maximum coverage spending the minimum cost possible or on a budget. Budgeted maximum Coverage problem seeks to find the maximum utility or profit from a resource based on a budget. In this thesis, we empirically analyze the exact algorithm in [1]. We implement the algorithm and test it on random, trees and binary trees analyzing running time and space requirements.

The rest of this chapter provides basic definitions on covering problems. It covers the definition of the budgeted maximum coverage problem and some terminologies related to approximation algorithms. This chapter concludes with previous work related to budgeted maximum coverage problem.

1.1 Covering Problems

Covering problems are computational problems that ask whether a certain combinatorial structure covers another, or how large the structure has to be to do that.

They can be formatted as minimization problems and integer programs. The most common example of covering problems is the SET COVER problem. Given a collection of sets $S_1, S_2, ..., S_n$ whose union is the universal set U, the optimization set cover problem is to find the minimum collection of the S sets whose union would be the universal set. For example: in Figure 1.1, there are six sets $\{A, B, C, D, E, F\}$. Their universal set U covers all the elements in $A - F$. However, the minimum number of sets needed to cover all elements is C and E.

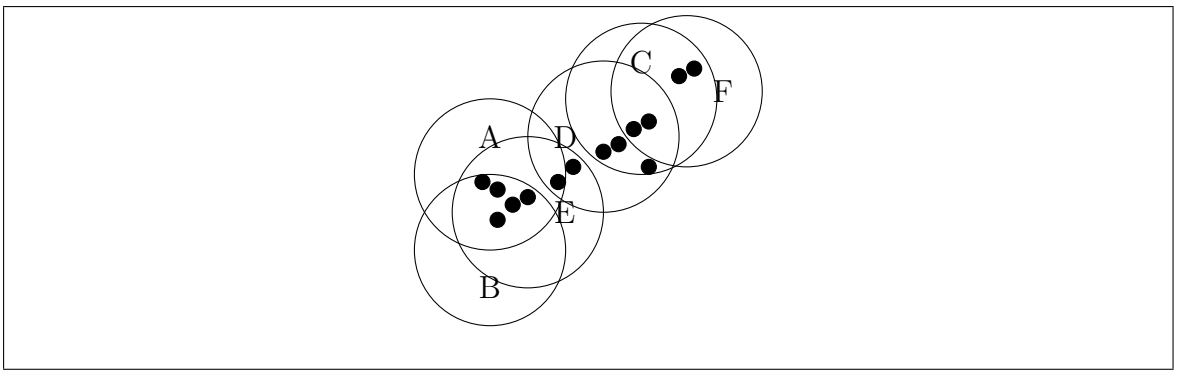


Figure 1.1: Minimum set cover is $\{C, E\}$.

Finding the optimal set cover is **NP-hard** [2].

1.2 Vertex Cover Problem

Given an undirected graph $G = (V, E)$, where V is the set of vertices $\{v_1, v_2, ..., v_n\}$ and E is the set of edges $\{e_1, e_2, ..., e_n\}$. The vertex cover problem on graph is to find a subset of vertices $S \subseteq V$ such that for every edge $E = (v_i, v_j)$ of the graph, either 'v_i' or 'v_j' is in S. A graph's vertex cover thus covers all edges of the graph. The vertex cover problem is one of the classical **NP-complete** problems [3]. The minimum vertex cover (MVC) is an optimization problem. MVC is **NP-hard** [4]. It seeks to compute the vertex cover with the minimum cardinality. In Figure 1.2, the graph has eight vertices. Some of the graph's vertex covers are $\{A, B, C, D\}$, $\{B, C, D\}$ and $\{A, E, F, G, H\}$. However, the minimum vertex cover is $\{B, C, D\}$.

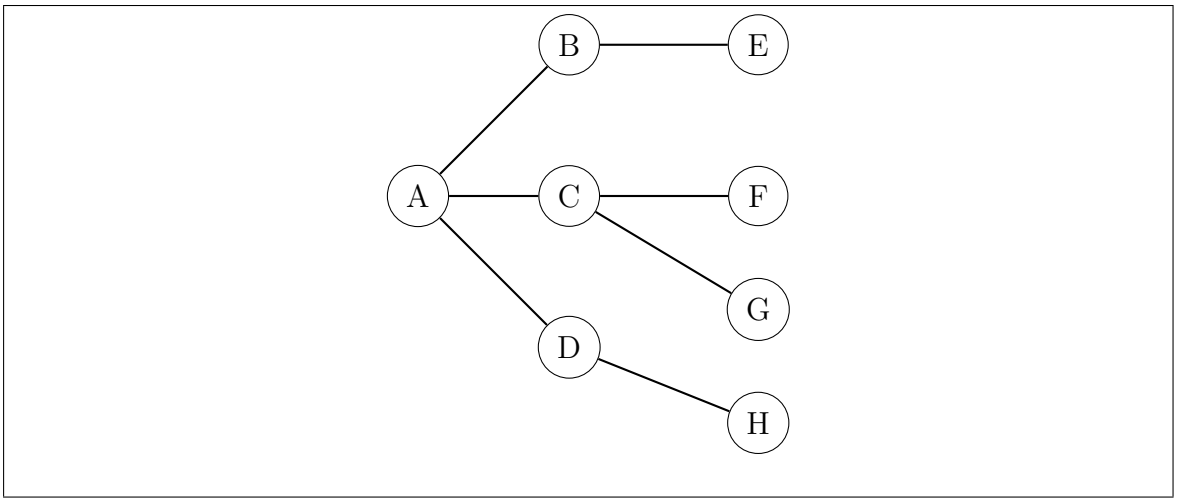


Figure 1.2: Minimum vertex cover is $\{B, C, D\}$.

1.3 Maximum Coverage Problem

The maximum coverage problem has three versions: the unweighted maximum coverage, weighted maximum coverage and budgeted maximum coverage. The definition of the maximum coverage problem is the same as that of the unweighted maximum coverage.

1. Unweighted maximum coverage(UMC): Given a set of sets $T = \{S_1, S_2, ..., S_n\}$, and a number K, select at most K sets $\{S_{c1}, S_{c2}, ...S_{ck}\}$ from T, such that the maximum number of elements is covered.

2. Weighted maximum coverage(WMC): WMC is identical to UMC except that every element in the sets have a weight. We want to find a coverage that has maximum weight.

3. Budgeted maximum coverage(BMC) : In BMC, each element has a weight and each set, a cost. A budget B is given instead of a number k. Pick the sets whose total cost $C \leq B$ gives the maximum weight.

1.4 Trees

A tree is a connected, undirected, acyclic graph [5]. It is rooted unless otherwise stated. Vertices and edges may be weighted or unweighted. An unweighted tree example is shown in Figure 1.4. The components of a rooted tree are as follows:

- root: The root is the first node of a tree. A root has no parents.
- vertex: is a node in the tree. It stores a data element.
- edge: is a link that connects two vertices.
- leaf: is a vertex with no child.
- centroid: is a vertex that splits the tree into two almost-equal halves.
- parent: A vertex's parent is the vertex that precedes it in hierarchy. Only the root node has no parents.

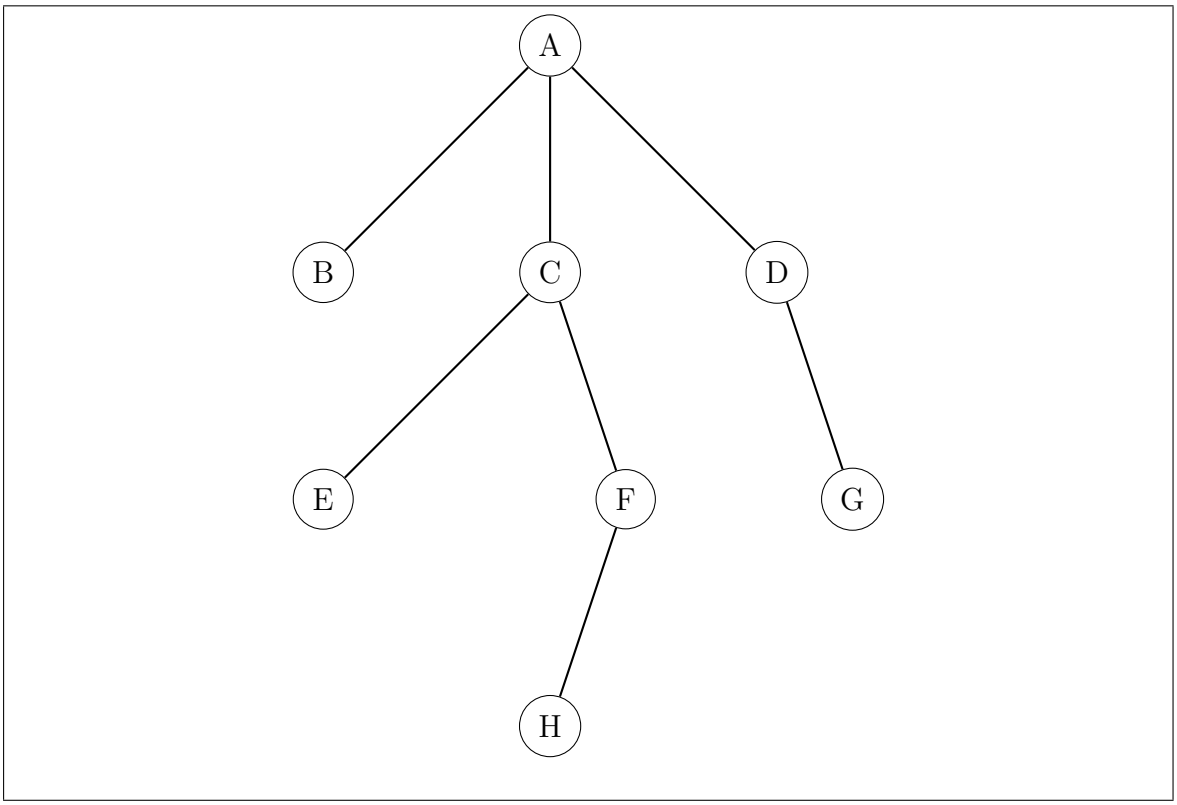


Figure 1.3: A tree

1.5 Budgeted Maximum Vertex Cover on Trees

A budget is a constraint on resources such as money, asset, time, and so on. The budgeted maximum vertex cover is finding the greatest cover based on a budget. It is a generalization of the maximum coverage problem. The Budgeted Maximum Vertex Cover on Trees (BMVCT) is defined as follows: Given a tree $T = (V, E)$, with every vertex $v \in V$ associated with a cost $c : V \to \mathbb{N}$, every edge associated with a profit $p : E \to \mathbb{N}$ and a budget B. The BMVCT problem is to find a subset of vertices with total cost $B' \leq B$ such that the total profit gained is maximized. We gain profit by covering edges. In Figure 1.5, we give an example BMVCT input and it's solution.

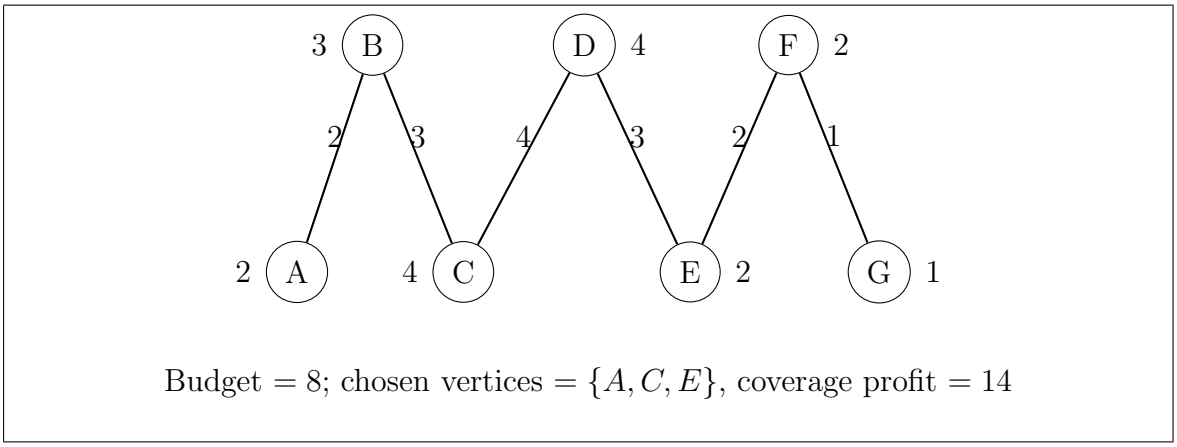


Figure 1.4: Budgeted maximum vertex cover on trees

1.6 Partial Vertex Cover Problem

The partial vertex cover problem (PVCP) is a generalization of the vertex cover problem. Given an undirected graph $G = (V, E)$ and an integer k, we wish to choose a minimum number of vertices such that at least k edges are covered. In Figure 1.6, we have an undirected graph with eleven vertices, if k is 3, the partial vertex cover (PVC) = $\{A\}$ or $\{C\}$; if k is 6, PVC = $\{A, C\}$.

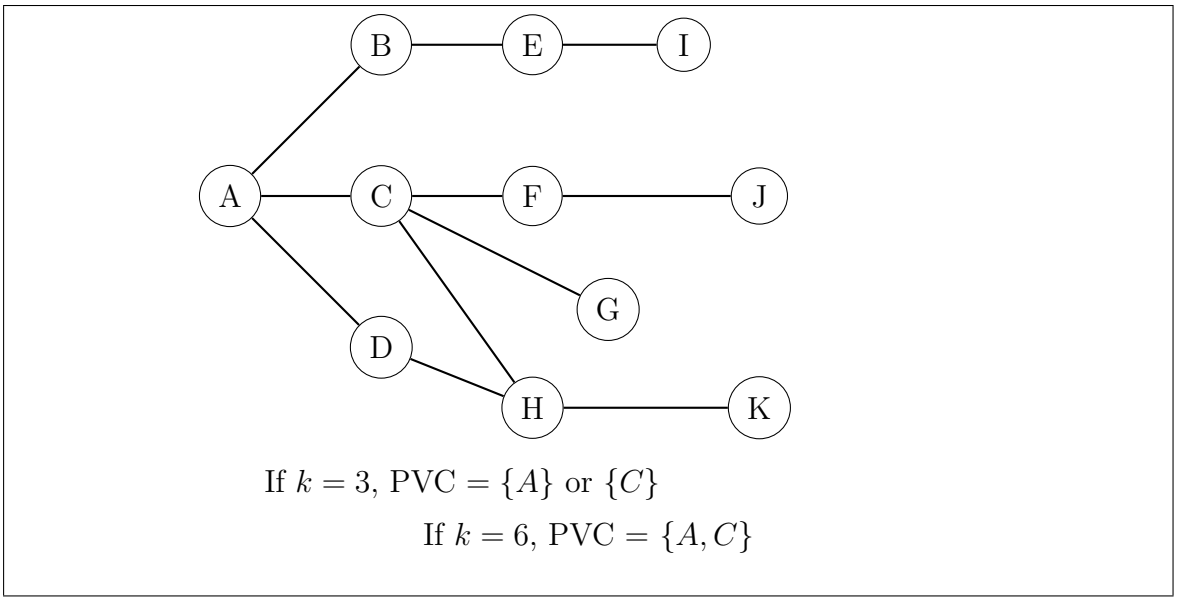


Figure 1.5: Partial Vertex Cover (PVC)

1.7 Algorithm-related Definitions

In this section, we define approximate and exact algorithms, PTAS and performance ratio.

- Approximation ratio/factor: In minimization problems, an approximation algorithm is said to have an approximation factor or ratio of δ, if for every instance of the problem, it gives a solution at most δ times the optimal value for that instance and $\delta \geq 1$. In maximization problems, an approximation algorithm is said to have an approximation factor or ratio of δ if for every instance of the problem, it gives a solution at least δ times the optimal value for that instance and $\delta \leq 1$ [6].

- Approximation Algorithms: are algorithms which give close answers to the optimal solution. They are efficient, polynomial-time algorithms that provide approximate solutions to **NP-hard** problems [7].

- Performance Ratio: Let A_s be the solution returned by a problem's approximate algorithm and O_s be the optimal solution to the problem. In minimization

problems, the performance ratio is the ratio of A_s to O_s. In maximization problem, it is the ratio of O_s to A_s [6].

- Exact Algorithms: These are algorithms that always provide the optimal solution to the input problem.

- Polynomial Time Approximation Scheme (PTAS): A family of algorithms is referred to as a polynomial time approximation scheme for an optimization problem. If for every $\epsilon > 0$, there is a $(1 + \epsilon)$- approximation algorithm in the family of algorithms for the optimization problem whose running time is polynomial for a fixed ϵ.

- Fully Polynomial time approximation scheme (FPTAS): A PTAS is a FPTAS if the approximation algorithm's running time is polynomial in both input size and $\frac{1}{\epsilon}$ [6].

The remainder chapters of this thesis is organized as follows: In Chapter 2, we discuss the reasons that motivated studying the budgeted maximum coverage problem on trees (BMVCT). Previous work related to BMVCT are also provided in Chapter 2. We cover some techniques for solving the budgeted maximum coverage, the approach we chose and why we chose the dynamic programming approach in Chapter 3. Chapter 4 covers implementation details, our empirical set-up and our results. We conclude this thesis and give some future work that can be done in Chapter 5.

Chapter 2

Motivation and Related Work

2.1 Motivation

The maximum coverage problem was used by Cornuejols *et al.*, [8] to model the problem of companies maximizing floats by locating bank accounts in multiple cities. The amount of days needed to clear a check drawn on a bank in city a depends on the city b where it is cashed. A company that pays bills to multiple clients in different locations can find it beneficial to maintain accounts in several strategically located banks in order to optimize its available funds. The company would then pay bills from the bank in city $b(a)$ with the highest clearing period to the clients in city b. The account location problem is modeled as follows: We are given the set of possible account locations $l_1, .., l_n$, the set of client locations $p_1, ..., p_m$,the cost of maintaining an account in a city c a_c, the numerical value of checks paid in city a, the amount of days to clear a check issued in a city c and cashed in city a k, the maximum number of accounts that can be maintained M. This problem is then solved as a maximum coverage problem aiming to maximize the delay in payments thus increasing available funds.

Maximum coverage can also be applied to decision support applications. Harinarayan *et al.,* [9] modeled databases as lattice frameworks, and designed algorithms to pick a good set of queries to precompute to optimized response time. Krause [10] applied maximum coverage in solving the placement and scheduling of sensors in

environmental monitoring, wireless sensor networks, and other applications. Kempe et al., [11] used maximum coverage in the problem of selecting the most influential nodes in a social network for marketing purposes and so on. In Kempe et al. [11], they used co-authorship in physics publication to build a collaboration graph. For every paper that had more than one author, an edge was inserted in the graph for each pair of authors. If two authors had co-authored more than one publication, it was inserted as multiple parallel edges in the graph. The graph was then solved as a coverage problem to pick the nodes that provided maximum coverage.

Maximum coverage is also used in modeling information retrieval problems [12], [13] and [14]. Maximum coverage is used to solve many resource allocation problems in communication network and information systems [15]. Hochbaum and Pathria [16] gave several applications of the maximum coverage problem. Maximum coverage problem can be used to solve packing and layout problems such as VLSI design, in circuit layout and design, in scheduling and in logistics [16]. In the scheduling problem, we are given a set of jobs $J_1, ...J_k$, a set of machines $M_1, ..., M_n$ each having some constraints on job groupings, the aim is to have the optimum weight set of jobs scheduled for the n machines. A logistics problem example is loading n vehicles with the maximum weight collection of items to be transported to the same destination. Loulou [17] used maximum coverage in testing printed circuit board (PCB) components for short circuits. The PCB was modeled as a graph. Its connection points were represented as vertices while the components were regarded as edges. In a testing stage, all components to be tested for short circuits must belong to the same cut in the graph. The problem aimed to test as many components as possible in the minimum possible amount of stages.

Maximum coverage was used by [18] and [19] in the problem of locating discretionary service facilities such as automatic teller machines and gasoline service stations. Saha and Getoor [20] generalized maximum coverage in solving web streaming. The budgeted maximum coverage (BMC) problem is a generalization of the well studied maximum coverage problem. BMC is used in resource allocation. The resource allocation problem is assigning available resources to various uses to achieve

maximum value. BMC is used in facility location problems [21] and [22]. the problem is to place facilities optimally based on constraints. Example constraints include transportation cost, number of demand points that must be served, distance minimization from residence due to hazardous materials and so on. BMC is used in job scheduling. Each job has a start and finish time. The job scheduling problem is to get the maximum profit subset of jobs which do not overlap. BMC is used in cell planning to ensure adequate network coverage. It is used in cyber-security to prioritize investment in security mitigations and maximize coverage of vulnerabilities. BMC is used in implementing strategies to deploy road side units so their spatio-temporal coverage is maximized under a limited budget.

Coverage problems can be adapted to solve various problems. The budgeted maximum coverage (BMC) models many real world problems. We choose to study BMC in trees because they can be efficiently analyzed. Given the importance of the BMC problem and its widespread use, this thesis seeks to implement the dynamic programming algorithm in [1] to verify their running time and effectiveness for BMVCT.

2.2 Related work

Ageev and Sviridenko [23] designed an approximation algorithm with a factor of $(1-(1-\frac{1}{k})^k)$ for the maximum coverage problem where k is the maximum size of subsets that are covered. They used a deterministic procedure of rounding linear relaxations to obtain this ratio. Khuller et al., [24] presented an approximate algorithm with a factor of $(1-\frac{1}{e})$ for the budgeted maximum coverage problem in undirected graphs. They proved that the natural greedy approach which picked the next lowest cost vertex was ineffective. They modified the condition for the greedy algorithm. Their greedy heuristics algorithm gave as output the candidates with the maximum weights. Their condition was to pick the next vertex with the greatest weight to cost ratio. Enumeration technique was then applied to all candidate solutions. The best solution was then chosen.

Apollonio and Simeone [25], Caskurlu et al., [26], and Joret and Vetta [27] estab-

lished the hardness of the Budgeted Maximum Coverage (BMC) in bipartite graphs. Apollonio and Simeone [28] studied the maximum vertex coverage (MVC) on bipartite graphs.They used the structure of the fractional solution of a linear programming formulation of MVC. They provided a greedy rounding approximation algorithm whose guarantee is $\frac{4}{5}$ times the optimum value of MVC on bipartite graphs. Caskurlu et al., [26] used linear programming relaxation to achieve an approximation algorithm with a factor of $\frac{8}{9}$ for BMC in bipartite graphs. Cohen and Katzir [29] defined the Generalized Maximum Coverage Problem (GMCP). GMCP is a generalization of the BMC problem.The GMCP problem is as follows: given a set of elements e each with a weight and profit both positive, a collection of bins b with individual weights, and a budget. GMCP's goal is to find the selection triple with maximum profit whose weight is less than the given budget. Cohen and Katzir [29] used a variation of the greedy algorithm to create an approximation algorithm with a factor of $(\frac{(2e-1)}{(e-1)} + \epsilon)$. They reduced this factor by using partial enumeration to achieve a ratio of $(\frac{e}{(e-1)} + \epsilon)$. Kar et al., [30] formulated the budgeted maximum coverage optimization problem into a path coverage (Pcoverage) problem in partially deployed software networks. Pcoverage problem entails covering the paths of an entire network. Kar et al. [30], developed an heuristic algorithm for the Pcoverage problem (MUcPF). Their results showed that MUcPF was more consistent, more economical and had better Software Defined Network effect compared to previous path coverage algorithms. Zheng et al. [31], demonstrated an algorithm for the budgeted maximum multiple coverage (BMMC) problem with a factor of $(1 - \frac{1}{e})$. They used multiple coverage to reflect the implementation of a layered defense and formulated them as BMMC models. They achieved their approximation factor by designing greedy approximation algorithms to identify near-optimal solutions for the BMMC models. They also presented an approximation algorithm with a factor of $\frac{1}{2}$ for a variation of BMMC that used cardinality constraint and group cardinality constraints. Markou et al. [32] presented an approximation algorithm with a factor of 0.316 for the budgeted coverage of a maximum part of a polygonal area problem. Their algorithms used a greedy technique based on the budgeted maximum coverage. Zhang et al. [33], considered the

maximum coverage problem with group budget constraints (MCGBC). MCGBC is a generalization of the budgeted maximum coverage problem. Zhang et al. [33], used a greedy algorithm to provide an approximation algorithm with a factor of $(1 - e^{-1})$. Rawitz and Rosén [34] studied a variant of BMC, the online budgeted maximum coverage(OBMC). In OBMC, each set has a cost. The sets arrive one after another, a choice must be made to accept or reject set. Accepted sets can be dropped later. Rejected sets can no longer be picked. OBMC's goal is to maximize the total weight of the elements covered by the collection of chosen sets. The constraint is cost of chosen sets must be less than or equal to budget). They presented a deterministic $(\frac{4}{1-r})$ competitive algorithm for OBMC. 'r' is the maximum ratio between a set's cost and the total budget. Their algorithm used a greedy rule to determine if a set would be chosen or not.

Curtis et al., [35] studied a type of BMC problem, the budgeted maximum coverage with overlapping costs (BMOC). The problem is similar to BMC except that in BMOC, the cost of a set might overlap the cost of another set. They identified a feature of their data (an overlap condition). They the modified the greedy algorithm and enumeration technique in [24] to achieve an approximation algorithm with a constant factor of $1 - \frac{1}{e}$.

Wang et al. [36], studied a variant of BMC, budgeted cell planning for cellular networks with small cells (BCP). BCP problem aims "to maximize the number of traffic demand nodes whose required rates are fully satisfied with a given budget" [36]. They generalized the method in [24] to develop an approximation algorithm with a factor of $(\frac{e-1}{2e})$ for BCP. Manurangsi [37] provided an improved approximation algorithm with a factor of 0.92 for the maximum k-vertex cover in graphs. The max k-vertex cover problem is to determine the k vertices that provide maximum coverage of edges on an input graph. The edge-weighted version places weights on edges. It seeks to find the k vertices that maximizes total weight covered by edges.Manurangsi [37] presented a FPT approximation scheme using approximate kernels. His FPTAS algorithm runs in $(\frac{1}{\epsilon})^{O(k)}poly(n)$ time for the max k vertex cover problem in graphs. Pachos [38] developed a polynomial time approximation schema (PTAS) for the edge-

weighted version of the max k-vertex cover problem in bipartite graphs. The PTAS improved results obtained in [23], [39] and [37]. Mkrtchyan et al., [1] proved the hardness of the budgeted maximum vertex cover on trees (BMVCT). They used dynamic programming to provide an exact algorithm for BMVCT. The DP algorithm runs in $O(B^2 \cdot |V|)$ time.

To the best of our knowledge, the implementation of the budgeted maximum coverage on trees has not been done or published by anyone previously.

Chapter 3

A Dynamic Programming-based Algorithm

This chapter discusses the different approaches that can be used to solve the BMVCT problem. We discuss the divide and conquer approach and the dynamic programming in depth. We state the reasons for implementing the dynamic programming approach.

3.1 Techniques for Solving BMVCT

BMVCT input instances can be solved using the following techniques

1. Enumeration Approach: The enumeration technique lists all possible solutions to the input instance and chooses the best solution. This approach gives an exact solution to the input instance. However, it is not feasible for very large solutions as it runs in exponential time.

2. Greedy Approach: The greedy approach builds the solution vertex by vertex. It takes the next lowest cost vertex solution until budget is exhausted. This approach gives an approximate solution. Khuller et al., [24] used the greedy algorithm with a modified condition. They used a vertex's $\frac{weight}{cost}$ ratio to pick the next best vertex.

3. Divide and Conquer Approach: This approach divides the BMVCT input instance into subproblems and solves these subproblems recursively. It combines the solution for the subproblems to solve the BMVCT problem. It is extensively discussed in the next section.

4. Dynamic Programming (DP) Approach : This approach is used for problems with optimal substructure and overlapping subproblems. Intuitively, the dynamic programming approach does not work for BMVCT on graph due to overlapping costs. An edge can be covered by two vertices but can only be counted once. We overcome this problem by keeping track of the chosen edges. The DP approach is discussed more in Section 3.4.

3.2 Divide and Conquer Approach

The divide and conquer approach takes in a BMVCT input tree instance as input. It computes the solution to the input instance by splitting it into sub-problems. It makes use of a novel centroid decomposition method to split the tree instance. It then solves these subproblems recursively and optimally. The centroid decomposition allows a tree to be partitioned in two almost-equal, edge-disjoint sub-trees. Each tree partition has between one-third to two-third of the original tree. Also, the two sub-trees share an intersection vertex. The divide and conquer approach then returns the optimal solution to the tree. The divide and conquer approach has two cases.

- Base case ($E = 1$): The base case occurs when the tree has only one edge E_i. In this instance, the tree can no longer be divided. We compute the optimal solution for the tree by trying the four vertex set possibilities. We look at all edges adjacent to a vertex when computing the optimal solution.

    1. No vertex is covered: If budget is less than the cost of both vertices, we do not pick any vertex as solution. Divide and Conquer approach returns null.

2. Vertex U is covered: Divide and conquer approach returns total profit of all edges adjacent to U which have not been covered. If U is the optimal solution. If the cost of vertex U, $(c(U))$ is less than the budget, and cost of vertex V, $(c(V))$ is greater than the budget.

3. Vertex V is covered: Divide and conquer approach returns total profit of all edges adjacent to V which have not been covered. If V is the optimal solution. If cost of vertex U is greater than budget and cost of vertex V less than budget.

4. Both vertices are covered: If the total cost of vertex U and V is less than budget. Divide and Conquer approach returns total profit of all edges adjacent to U and V which have not been covered.

- Recurring case ($E \geq 2$): While the tree has more than one edge; the Divide and Conquer approach uses centroid decomposition to split the tree T into two sub-trees, T_1 and T_2. The intersection vertex of both trees is denoted by r. The Divide and Conquer approach computes two instances I_1 and I_2 with both trees and r. We loop for values of B from zero to B computing the two instances below.

  1. Instance 1 (I_1): r is not an element of the optimal set U^*. The divide and conquer approach recurses on T_1 with a budget B^* and on T_2 with budget $B - B^*$. Divide and Conquer approach stores the information that r has not been picked and its adjacent edges have not been covered.

  2. Instance 2 (I_2): r is an element of the optimal set U^*. Divide and Conquer approach recurses on T_1 with budget B^* and T_2 with budget $B - B^* - c(r)$. $c(r)$ is the cost of r. Divide and Conquer approach stores information that r has been picked and its adjacent edges covered.

The divide and conquer approach recurses B times every step, $O(B)$. The depth of the recursion is $O(\log n)$ with node degree $O(B)$. It follows that divide and conquer

approach runs in $((n \cdot B)^{O(\log n)})$ time. An instance of BMVCT is solved using the divide and conquer approach in Figure 3.1 with a budget of 4.

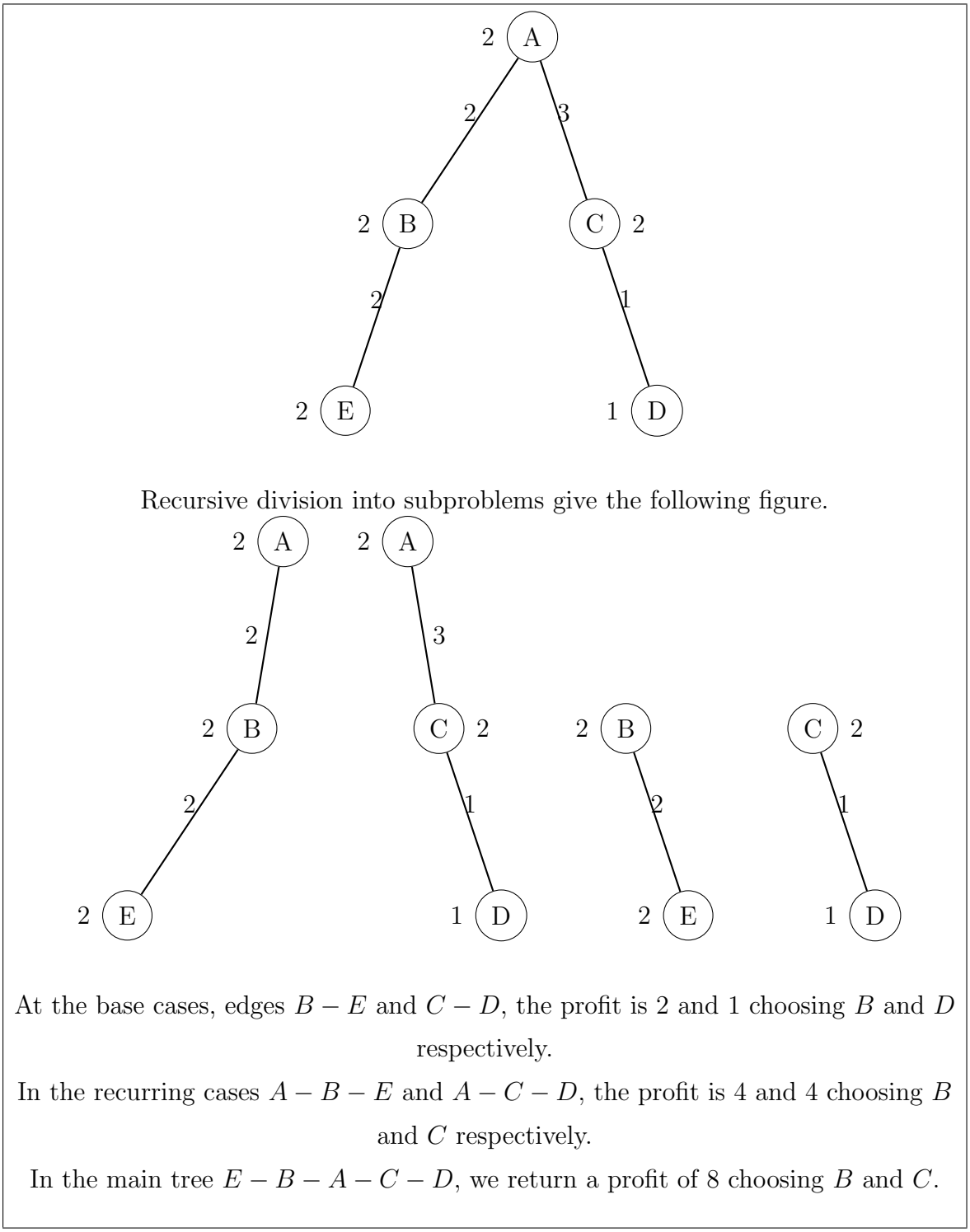


Figure 3.1: Using divide and conquer approach to solve a BMVCT instance

3.3 Dynamic Programming Approach

The divide and conquer algorithm computes solutions to sub-problems repeatedly wasting time and space resources. The BMVCT problem instance has an optimal sub-structure and can be solved by overlapping its sub-problems. The dynamic programming approach uses the idea of the divide and conquer approach however it saves the solutions to the sub-problems. The dynamic programming (DP) approach eliminates re-calculations by storing previously computed solutions in tables. It performs a centroid decomposition of the BMVCT tree input and stores the result in a centroid array. The DP approach then computes the solution upward from the leaves to the centroids using the centroid array. It uses the solution from the previous computation to solve the next level up the recursion tree. The dynamic program state has the following parameters:

1. Subtree(s) $S \subseteq T$.
2. Remaining budget $B_s \geq 0$.
3. U^+, the set of chosen vertices and U^- the set of vertices not chosen.

The dynamic programming approach runs in $O(B^2 \cdot |V|)$

3.4 Dynamic Programming Algorithm

The divide and conquer approach runs in quasi-polynomial time. We choose to implement the dynamic programming approach as it runs in polynomial time and gives an exact solution. Our dynamic programming (DP) algorithm takes as input a weighted tree, a centroid array, and a budget B. The DP algorithm creates a DP table to store computed solutions. It loops through the vertices in the array B times and stores the solution gotten at each iteration into the DP table. The DP algorithm uses a method to keep track of chosen edges so that the edges are not duplicated. The solution for the current iteration is gotten by taking the best choice either from

previous solution, the current computation or both current and previous computation. We illustrate our DP algorithm in the following example.

Given the tree below with a budget of 4:

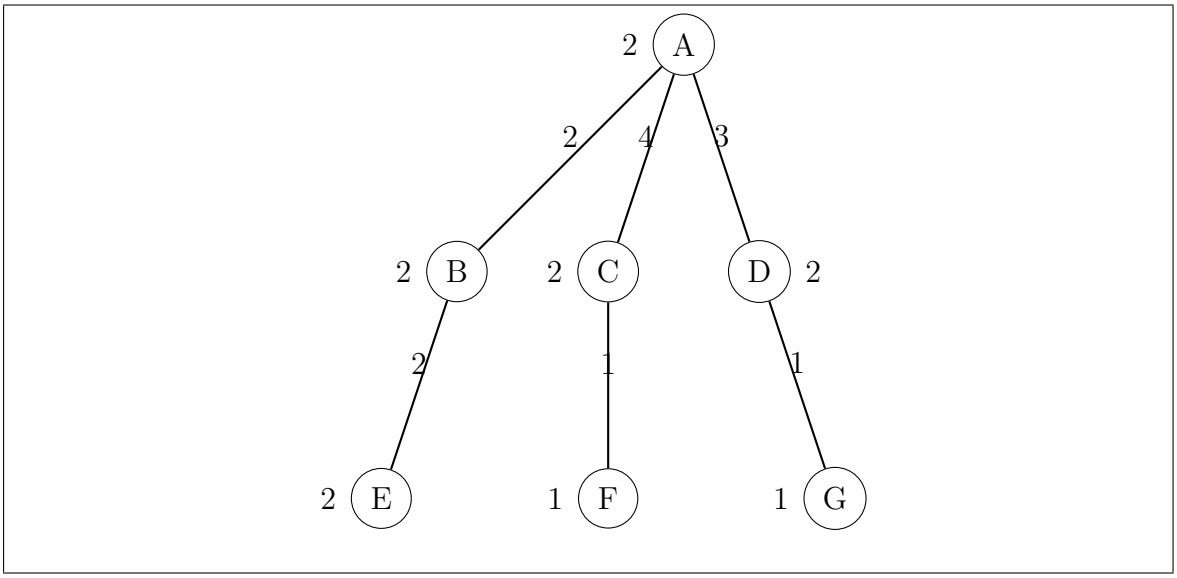


Figure 3.2: BMVCT tree

The DP algorithm solves the problem in the following steps:

1. The DP algorithm performs a centroid decomposition on the tree which yields a centroid array $C = \{E, B, F, C, G, D, A\}$ shown below.

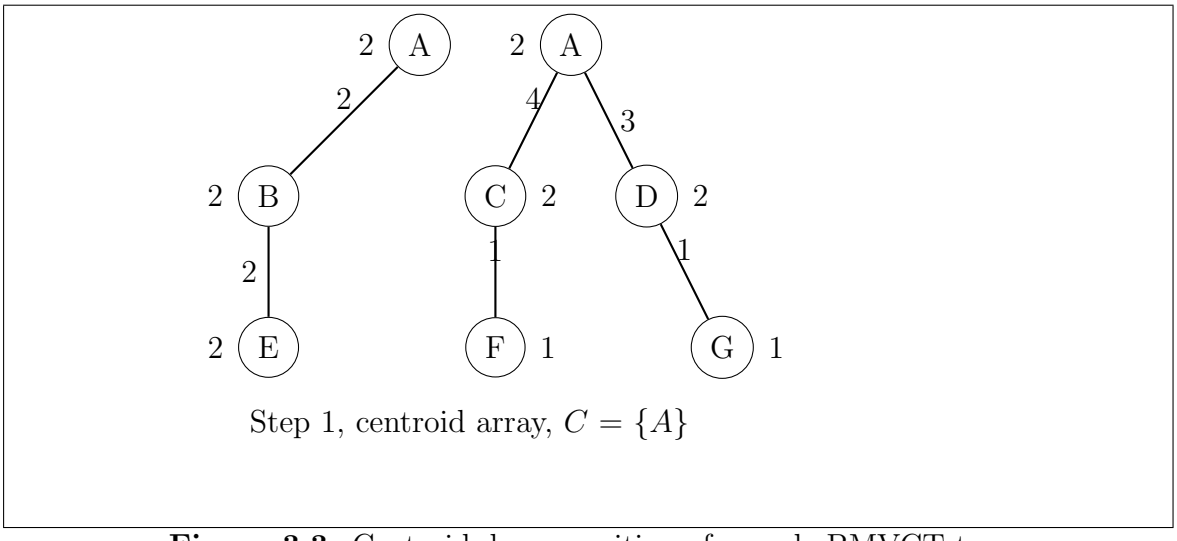
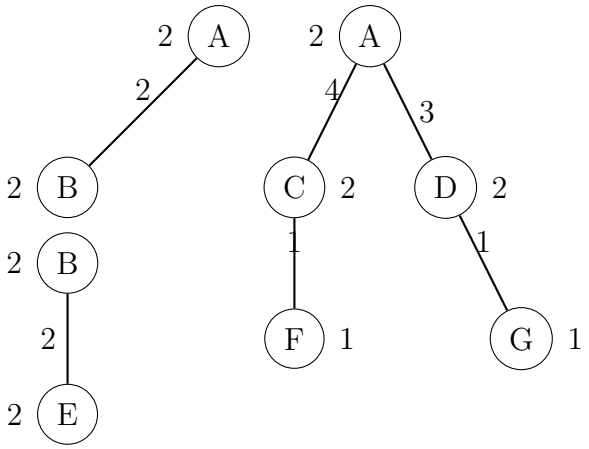
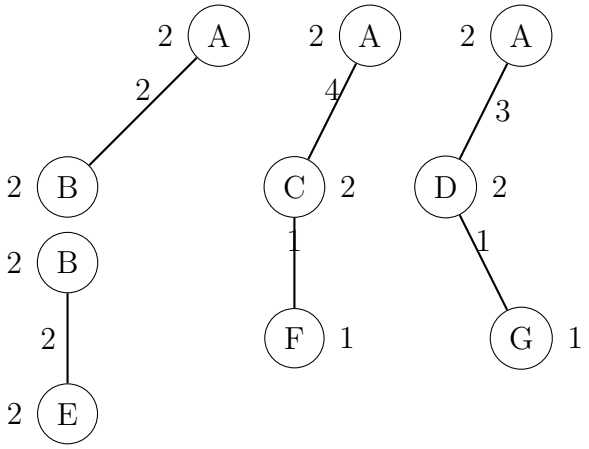


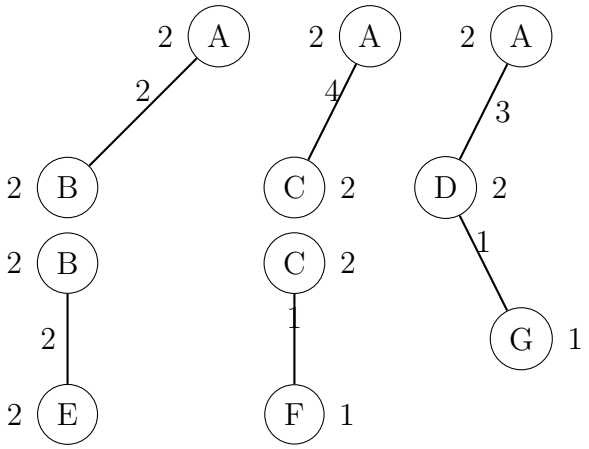
Figure 3.3: Centroid decomposition of sample BMVCT tree



Step 2, centroid array, $C = \{E, B, A\}$

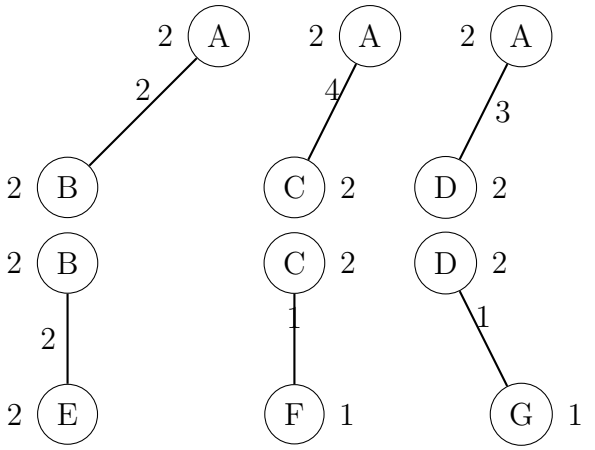


Step 3, centroid array, $C = \{E, B, A\}$



Step 4, centroid array, $C = \{F, C, E, B, A\}$

Figure 3.4: Centroid decomposition of sample BMVCT tree cont'd



Step 5, centroid array, $C = \{G, D, F, C, E, B, A\}$

The tree is fully decomposed in step 5.

Figure 3.5: Centroid decomposition of sample BMVCT tree cont'd

2. The algorithm iterates through the centroid array C, and values of budget B from 0 to 4 and fills the BMVCT-DP table. We use variable i to iterate through the vertices and variable j through the different budget values. The BMVCT-DP table is shown in Table 3.1. It is filled with the following steps.

   (a) if i or j is zero, entry $[i][j]$ is zero.

   (b) else if cost of vertex $c(i)$ is less than j(the current budget):

      i. calculate the profit $p(i)$ if the vertex is covered.

      ii. get the entry of the BMVCT-DP table at position $[i-1][j-c[i]]$.

      iii. add the profit of the entry to $p(i)$ and assign to variable $totalProfit$. A check is made to ensure no edge is duplicated.

      iv. get the entry of the BMVCT-DP table at position $[i-1][j]$ and assign it to variable $temp$.

      v. if $totalProfit$ is greater than $temp$, assign $totalProfit$ to BMVCT-DP table entry $[i][j]$'s profit and update covered edges.

      vi. else: if $totalProfit$ is less than or equal to temp, assign the values of BMVCT-DP table entry $[i-1][j]$ to BMVCT-DP table entry $[i][j]$, .

(c) else assign to BMVCT-DP table entry $[i][j]$, the values of BMVCT-DP table entry $[i-1][j]$.

Vertex	Budgets (0 through 4)				
i/j	0	1	2	3	4
0	0	0	0	0	0
G	0	1	1	1	1
D	0	1	4	4	4
F	0	1	4	5	5
C	0	1	5	6	9
E	0	1	5	6	9
B	0	1	5	6	9
A	0	1	9	10	11

Table 3.1: BMVCT-DP table for the input BMVCT tree

3. The algorithm returns the solution at BMVCT-DP table entry $[length(C)][B]$.

Chapter 4

Empirical Analysis

This section covers the BMVCT input instance, implementation details and setup. We round off the section by discussing the empirical results.

4.1 Input instance classification

This section discusses the different types of trees used and how they were generated.

4.1.1 Random trees

The random tree generator (RTG) takes the number of vertices as input. It generates a random tree of any structure with n vertices. It assigns cost weights to the vertices and profit weights to the edges. RTG randomly generates connection between vertices. It links the edge to both vertices to form a connected undirected tree. The random tree generated does not have any specific structure. An example of a random tree is shown in Figure 4.1.

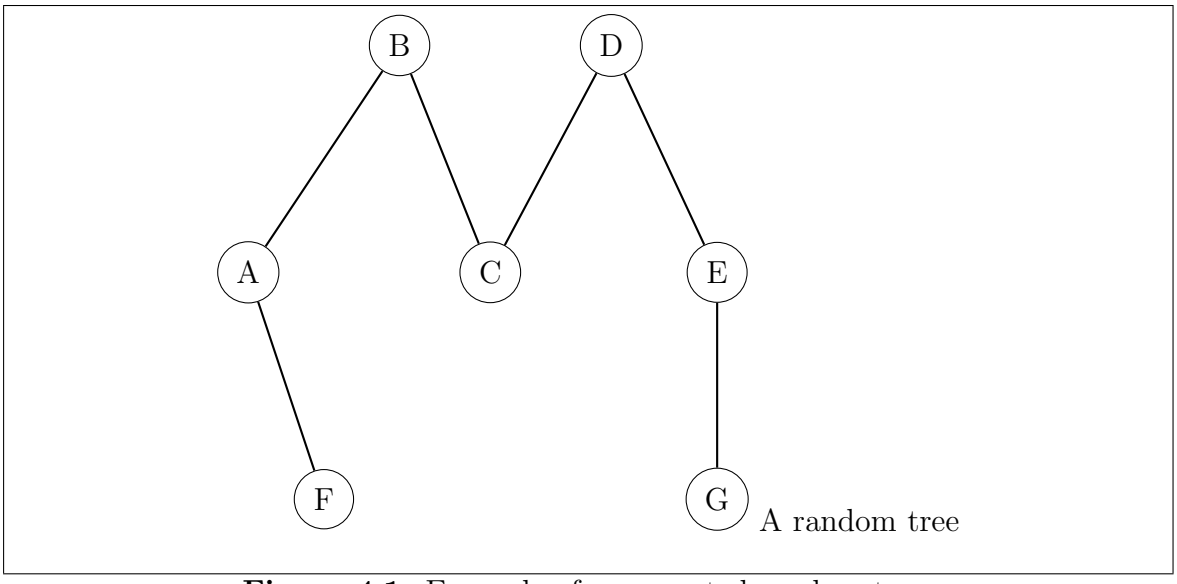


Figure 4.1: Example of a generated random tree.

4.1.2 Star trees

The star tree has one vertex at the center and every other vertex connected to it. The star tree generation is as follows:

1. Accept number of vertices n as input.
2. Iterate n times.
3. Make the first vertex central.
4. Assign weight to the central vertex.
5. Assign weight to the vertex being iterated on, i.
6. Connect i to the central vertex and vice-versa.
7. Assign weight to the edge connection.
8. When iteration is complete, return tree.

An example star tree is shown in Figure 4.2.

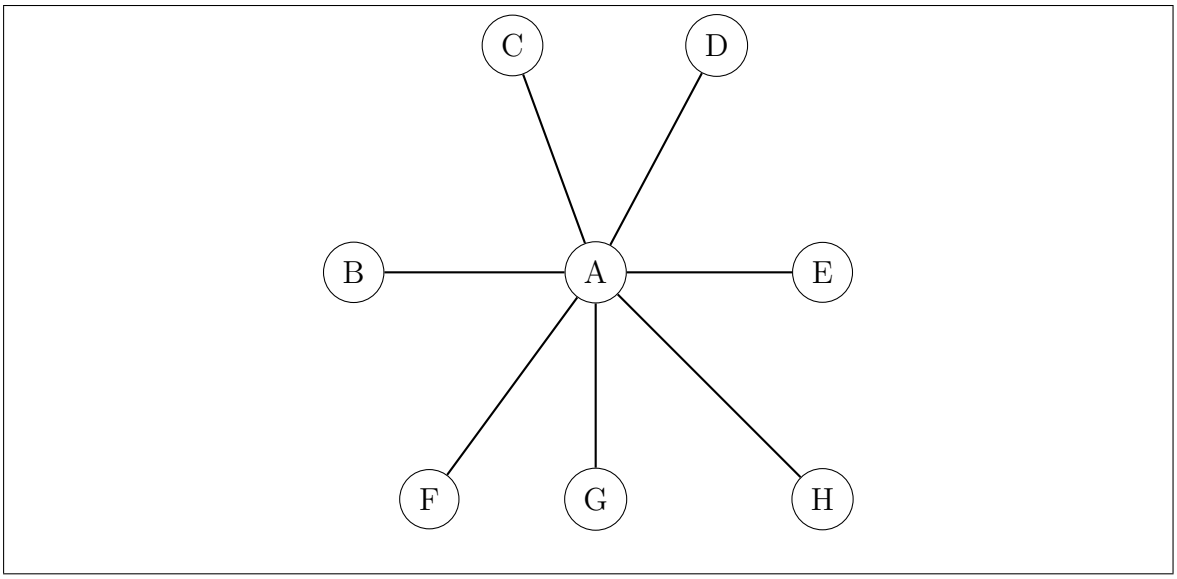


Figure 4.2: An example of a generated star tree.

4.1.3 Binary trees

The binary tree is rooted. Each vertex has zero, one or two children. A vertex is labeled i. If the node is odd, it's parent is located at $(int)(\frac{i}{2})$. If the node is even, it's parent is located at $(\frac{i}{2} - 1)$. A vertex's left child is located at position $(2 \cdot i) + 1$. A vertex's right child is located at position $(2 \cdot i) + 2$. The binary tree generator takes as input the number of vertices, n. The binary tree is then generated by iterating through n, assigning a vertex it's cost and parents. The parent-child connections are given edge weights. An example binary tree is shown in Figure **??**.

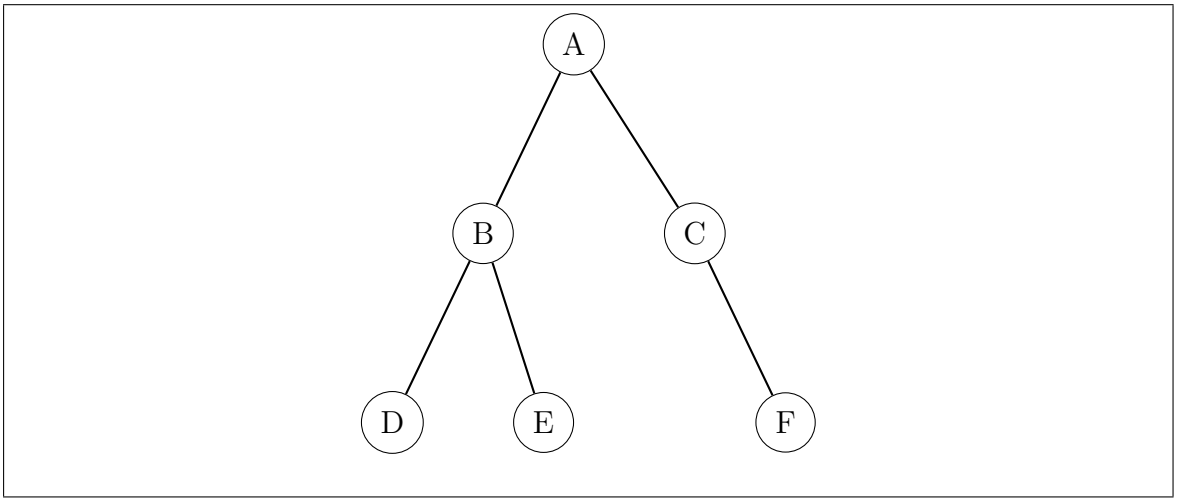


Figure 4.3: An example of a generated binary tree

4.2 Implementation Details

All algorithms were implemented in $C++$ with $C++17$ standard. The trees were represented as an adjacency list. Every vertex was represented as a *struct*. The adjacency list was an array of *struct*. Each vertex *struct* stored information on it's identifier, cost, the number of connected vertices, the identifiers of the connected vertices and connected edges. It also had information on the cost of connected edges. The tree adjacency list was implemented using array to enable fast access to positions for the centroid decomposition method and the BMVCT method. The connected vertices and connected edges were represented using linked lists to minimize space usage as they were not accessed very often. The centroid decomposition method also stored its results in an array for fast access to the decomposed tree. Optimal solutions to BMVCT subproblems were stored in *structs*. Each *struct* had information on the profit, the number of covered edges, the identifiers of the covered edges and the chosen vertices. This *struct* was also used for the variable used to keep track of covered edges to avoid duplication. The BMVCT method stored it's dynamic programming table in a two dimensional array. The dimensions of the array were the number of vertices and the budget. An array is used because of easier accessibility, easy assignment and compactness. The array stored the solution *structs*.

4.3 Empirical Setup

The implementation was compiled with *GNU GCC* compiler on *CodeBlocks*. The code was run on a Windows machine with intel core $i7$ processor. The machine had a 64-bit operating system, a 16 GB RAM and 2TB hard drive. Implementation was tested on random trees, star trees and binary trees of different sizes. The implementation's running time for the different trees and budget were obtained using *std::chrono* in the $C++$ library. The memory used by the implemented code was obtained using the "$GetProcessMemoryInfo$" method in the Microsoft Windows API, "$PSAPI$".

4.4 Empirical Results

This section presents the running times and memory used by our BMVCT implementation on the different trees used. We present the results for various tree sizes ranging from 5 to 1500 and budgets from 1 to 200. We also plot the graphs associated with it.

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
5	1	1180	7260
	3	1394	7260
	5	1952	7260
	10	2038	7264
	15	3231	7268
10	1	1779	7264
	3	1614	7264
	5	2020	7268
	10	2405	7272
	15	2367	7280
20	1	3895	7268
	3	5066	7272
	5	5983	7280
	10	5967	7292
	15	6970	7312

Continued on next page

Table 4.1 – continued from previous page

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
30	1	4449	7272
	3	6376	7272
	5	8196	7284
	10	9607	7316
	15	9804	7340
40	1	7184	7272
	3	9044	7280
	5	10672	7316
	10	12696	7348
	15	13512	7400
50	1	10786	7272
	3	11483	7316
	5	11728	7316
	10	14215	7376
	15	16693	7468

Table 4.1: Random tree running table 1

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
5	1	0	7260
	3	0	7260
	5	0	7264
	10	0	7268
	15	1562	7280
10	1	999	7264
	3	1846	7268
	5	2995	7284
	10	3997	7272
	15	4001	7284
20	1	1999	7268
	3	2963	7272
	5	6178	7288
	10	7826	7308
	15	8634	7332
50	1	6255	7288
	3	7621	7308
	5	9987	7352
	10	10031	7440
	15	12017	7548

Table 4.2: Star tree running-time table 1

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
5	1	211	7260
	3	756	7260
	5	1004	7264
	10	1251	7264
	15	1994	7268
10	1	2032	7264
	3	2995	7264
	5	3028	7268
	10	3125	7272
	15	3993	7280
20	1	3512	7268
	3	5992	7272
	5	6044	7276
	10	6915	7292
	15	7652	7316
50	1	9010	7272
	3	10018	7288
	5	10974	7308
	10	11995	7368
	15	13027	7444

Table 4.3: Binary tree running-time table 1

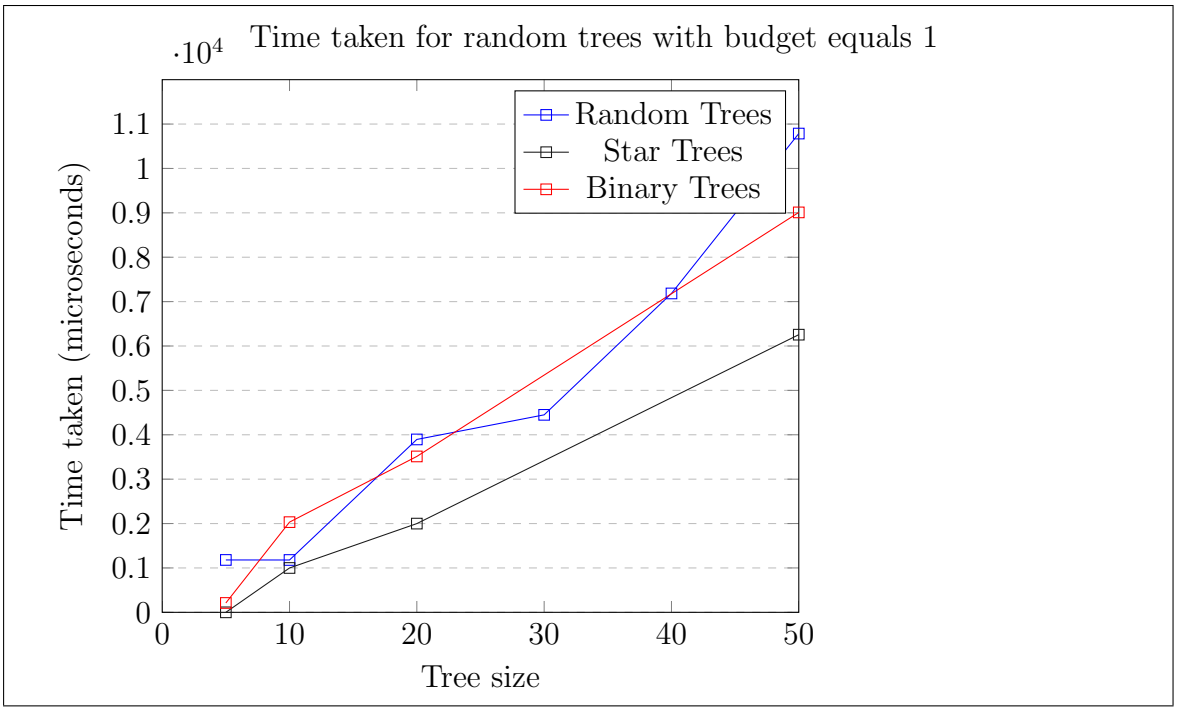


Figure 4.4: Random tree running time for budget equals 1.

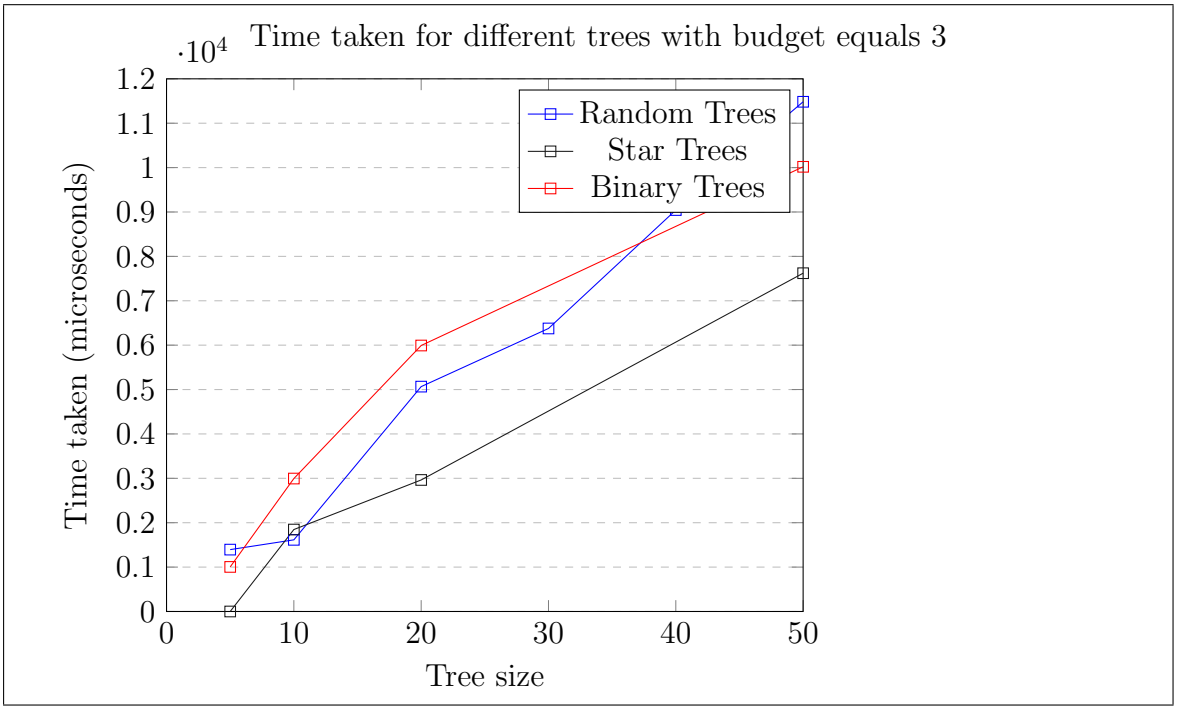


Figure 4.5: Different tree running times for budget equals 3.

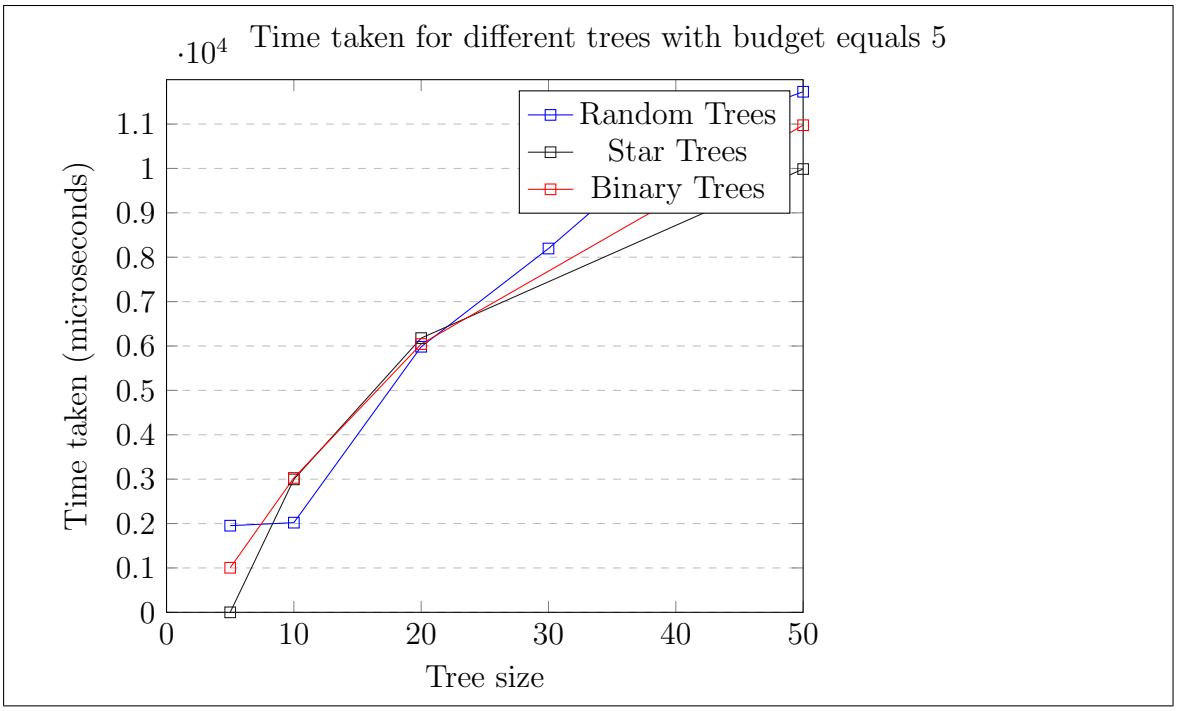


Figure 4.6: Different tree running times for budget equals 5.

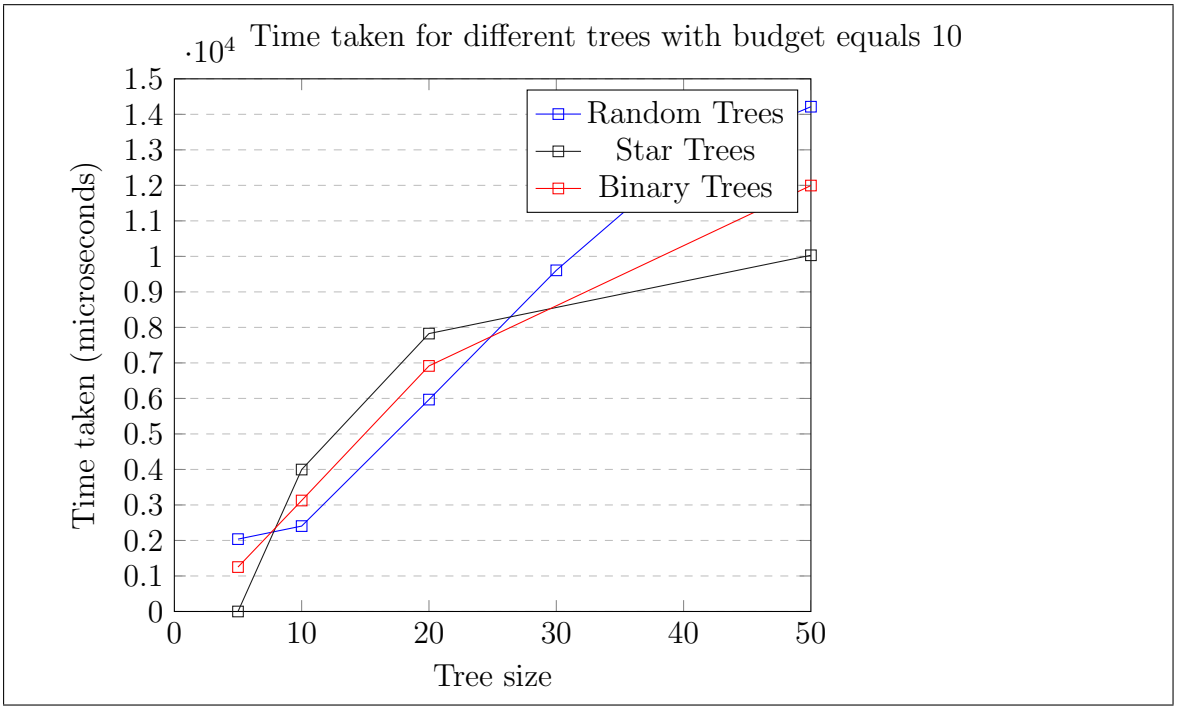


Figure 4.7: Different tree running time for budget equals 10.

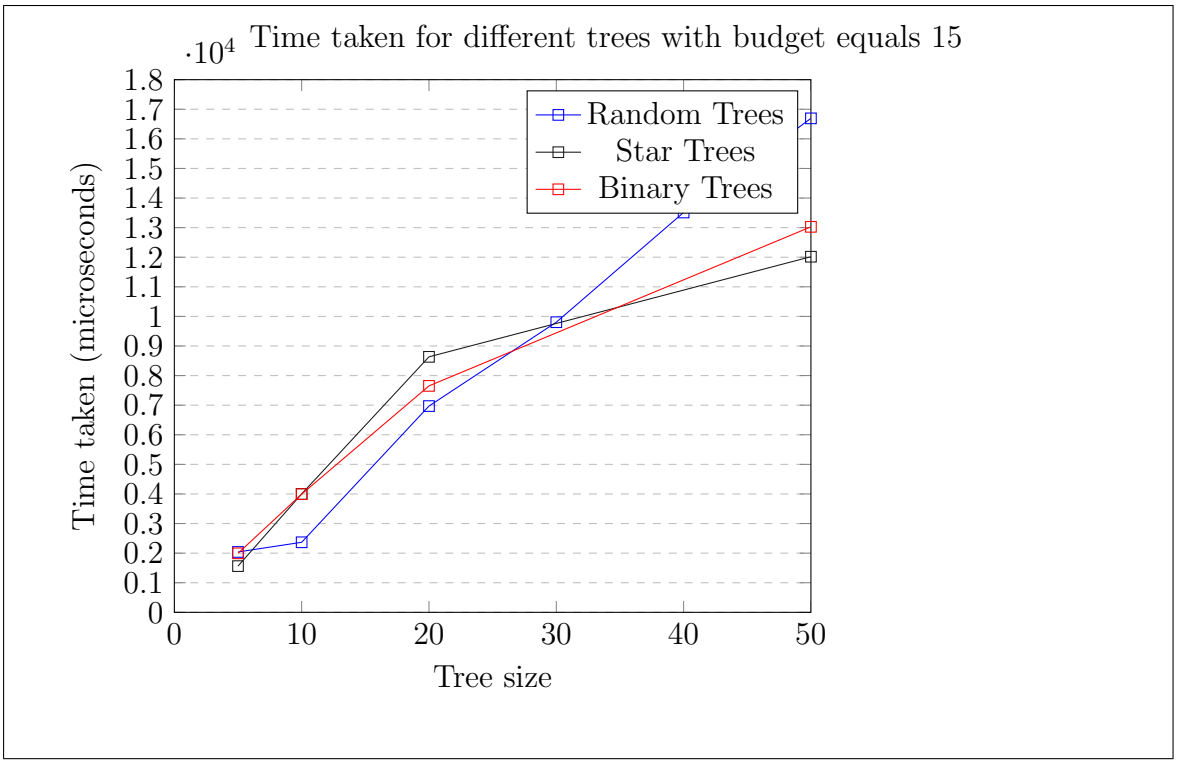


Figure 4.8: Different tree running times for budget equals 15.

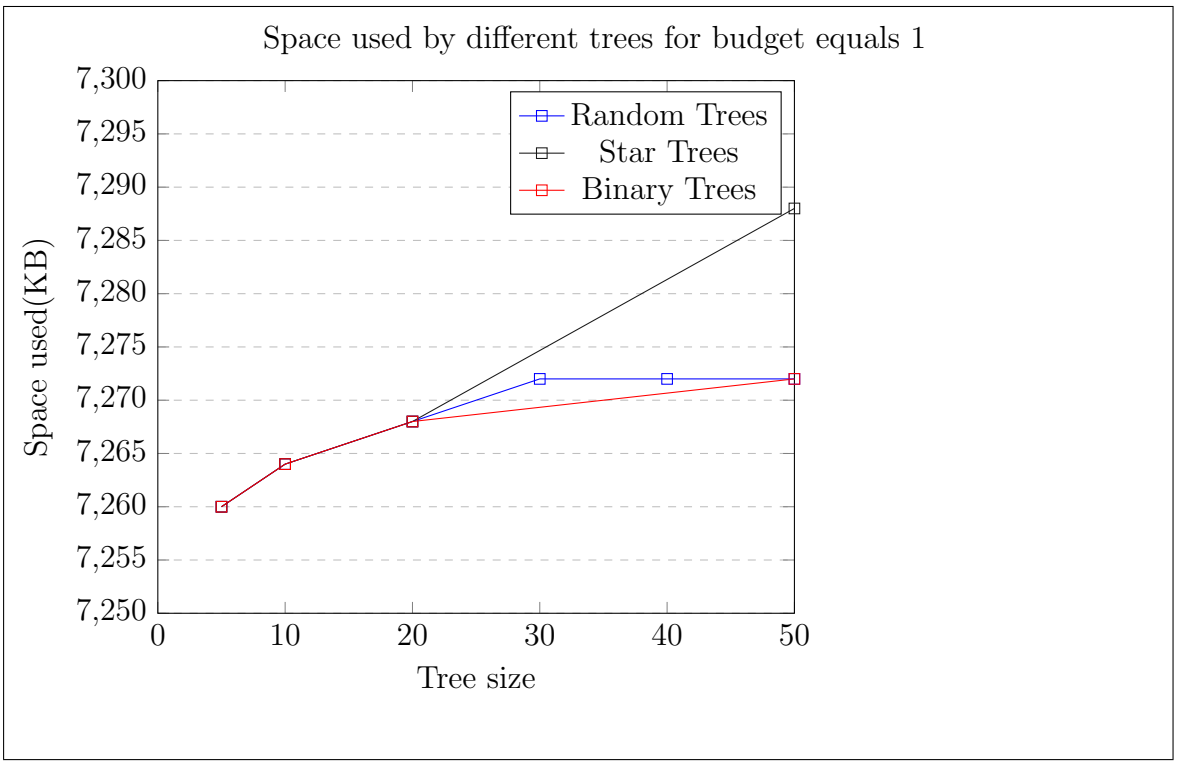


Figure 4.9: Space used by different trees for budget equals 1.

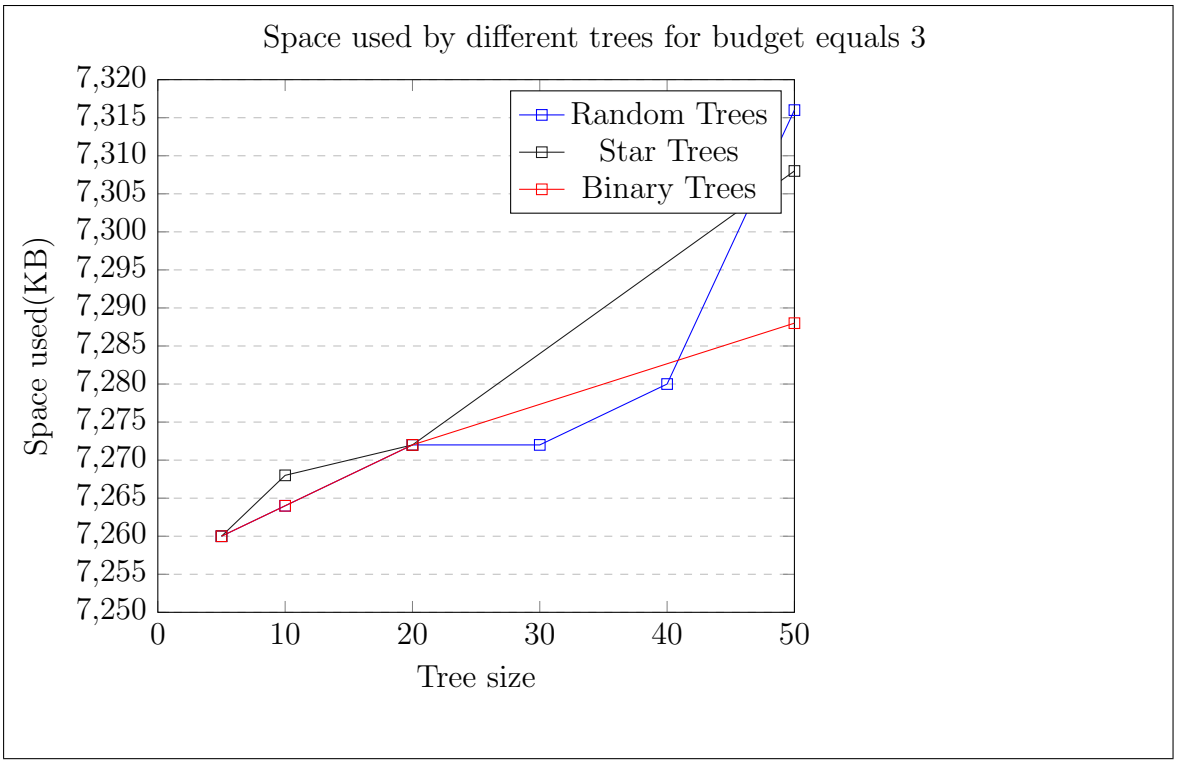


Figure 4.10: Space used by different trees for budget equals 3.

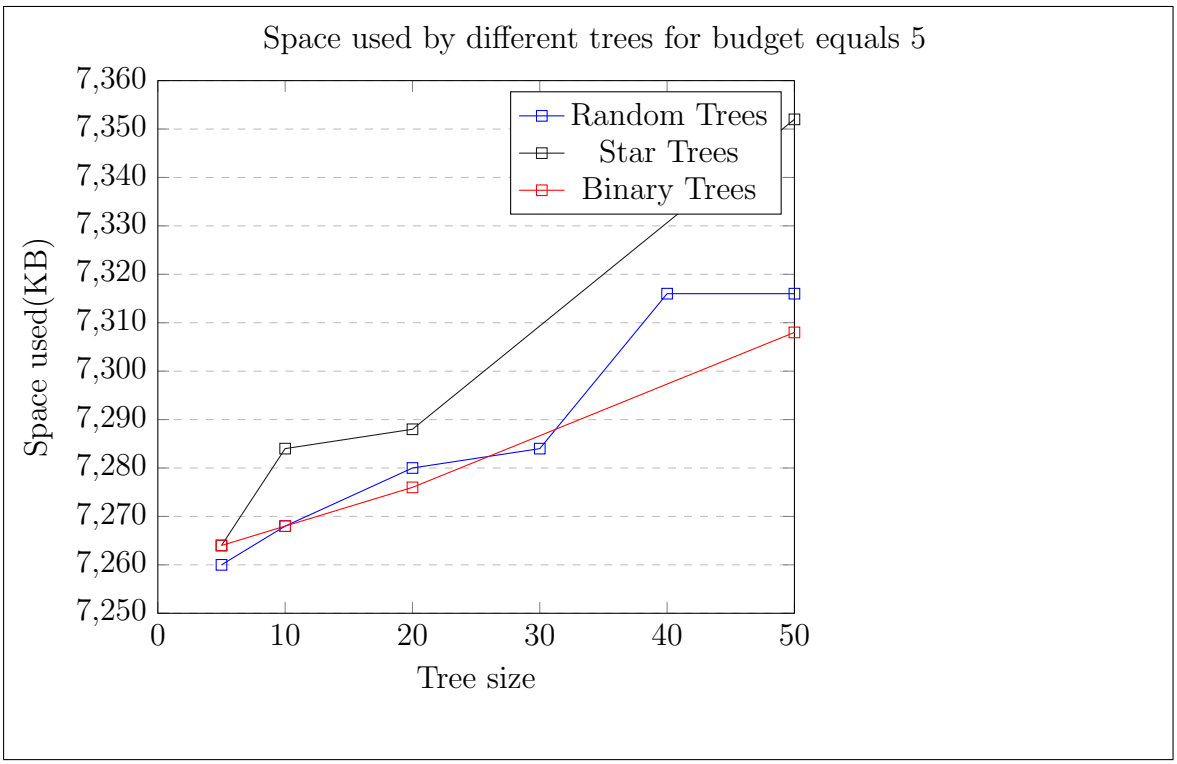


Figure 4.11: Space used by different trees for budget equals 5.

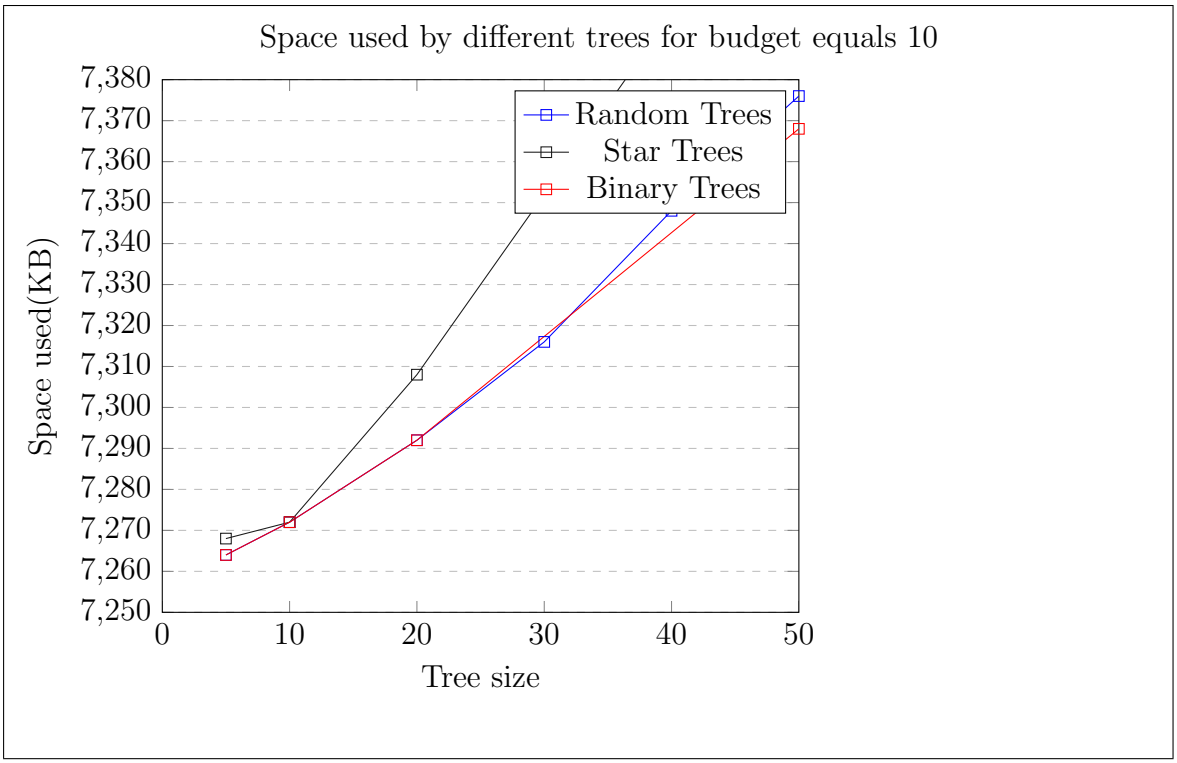


Figure 4.12: Space used by different trees for budget equals 10.

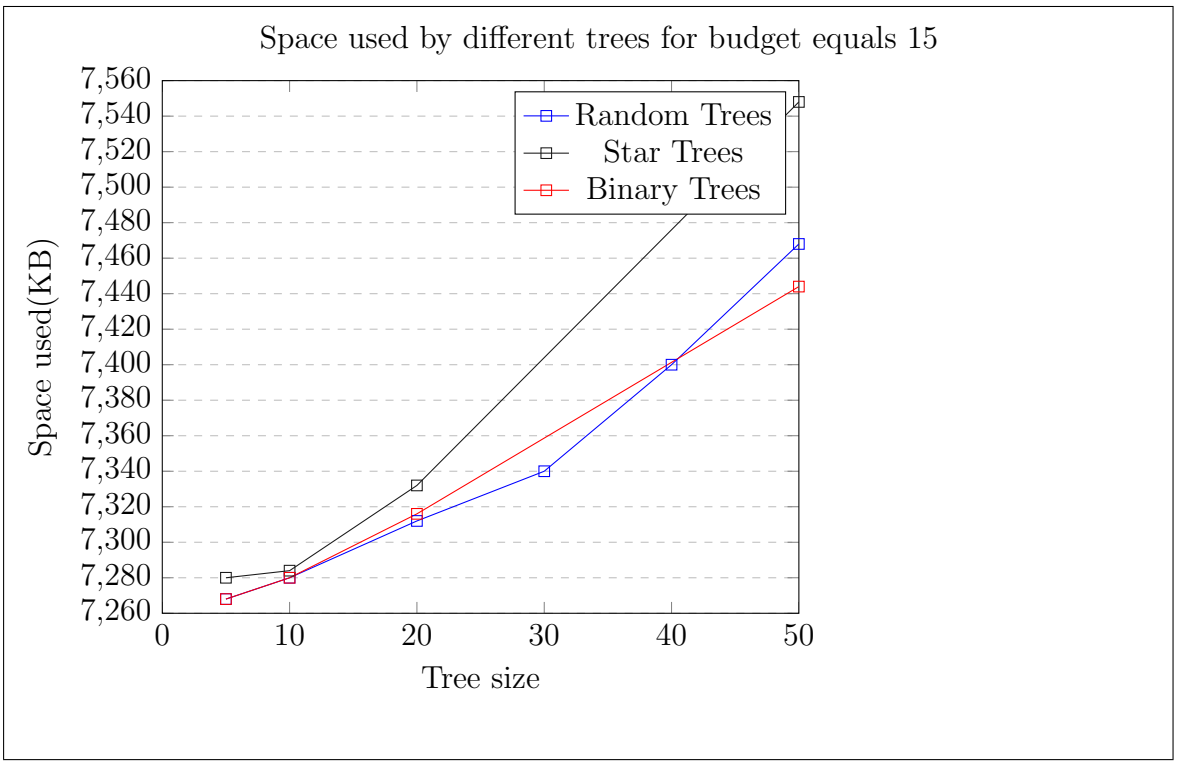


Figure 4.13: Space used by different trees for budget equals 15.

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
100	20	26513	7912
	50	48208	9960
	100	67992	13904
	150	86252	18876
	200	112478	23448
500	20	108416	11332
	50	190094	29296
	100	330637	70492
	150	500176	136740
	200	671020	208920
1000	20	214472	19664
	50	373729	54200
	100	743445	158856
	150	1098339	310096
	200	1537235	504180
1500	20	300776	25756
	50	569882	86832
	100	1117443	267840
	150	1766452	513128
	200	2535992	820308

Table 4.4: Random tree running-time table 2

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
100	20	31126	9024
	50	38963	11492
	100	39932	15640
	150	57906	19492
	200	73789	23884
500	20	116728	40428
	50	195418	105920
	100	336131	206440
	150	470725	306948
	200	609415	411252
1000	20	274269	146488
	50	540612	407664
	100	980380	807788
	150	1513124	1199460
	200	1928843	1645640
1500	20	526595	319976
	50	1033294	852096
	100	1968624	1787528
	150	2413562	2449059
	200	2895690	3061320

Table 4.5: Star tree running-time table 2

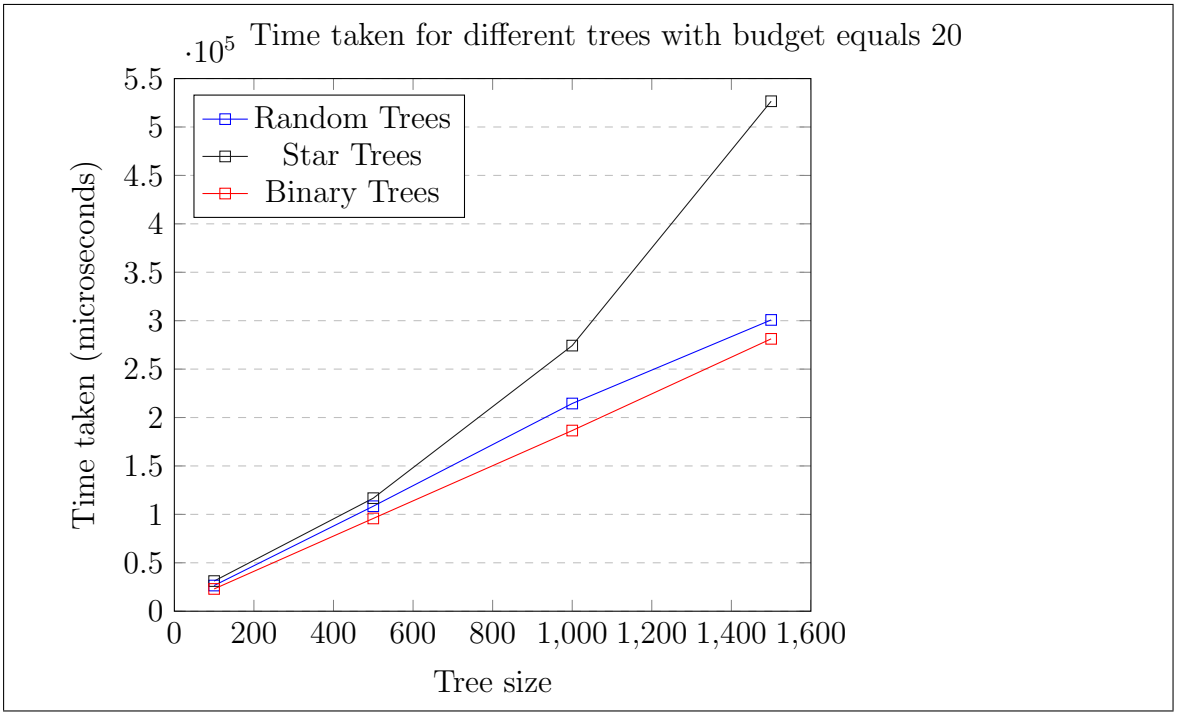


Figure 4.14: Different trees: running times for budget equals 20.

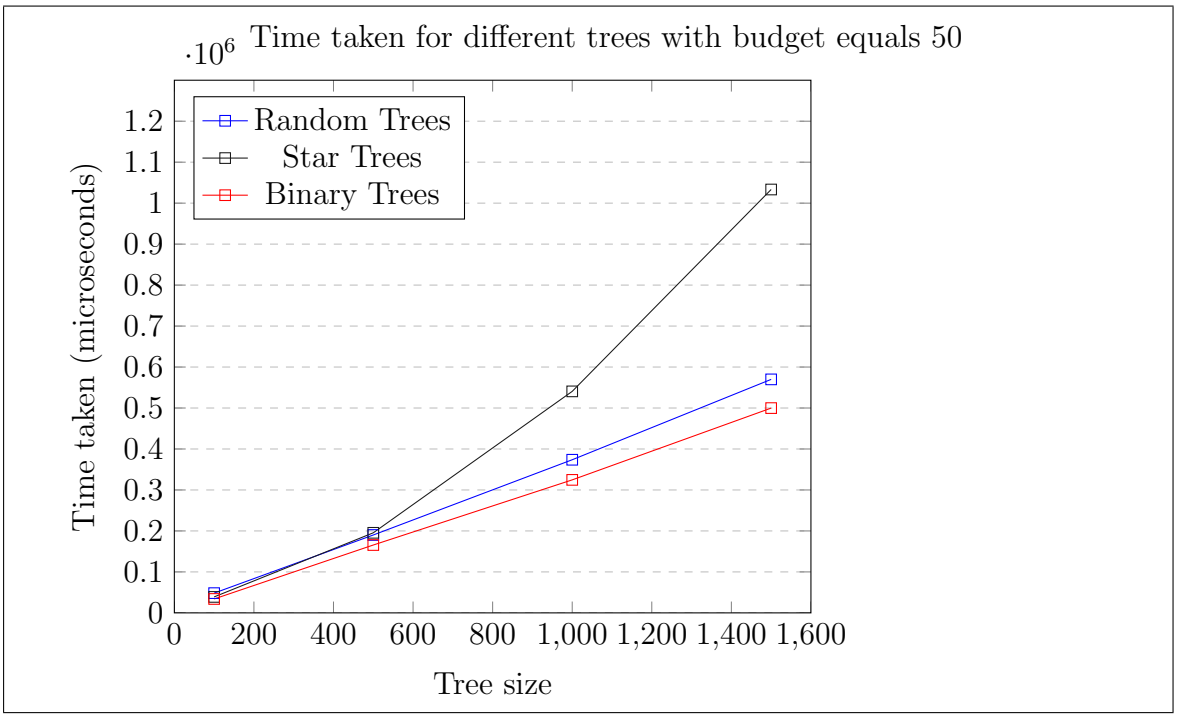


Figure 4.15: Different trees: running times for budget equals 50.

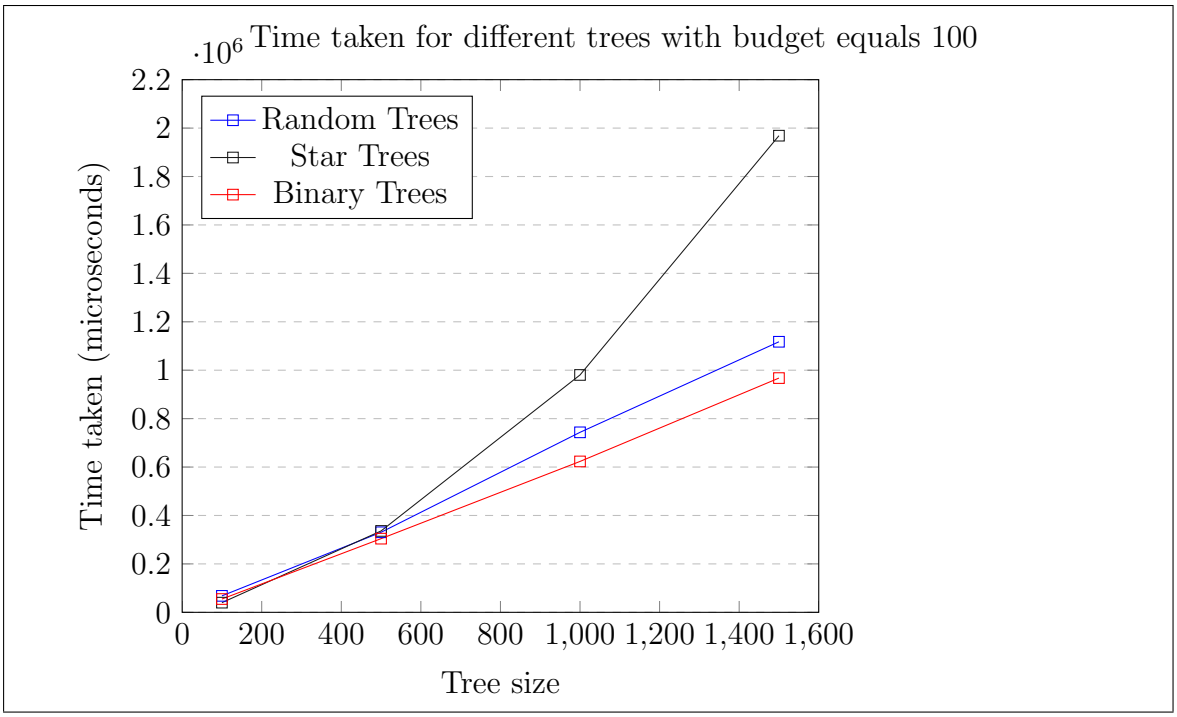


Figure 4.16: Different trees: running times for budget equals 100.

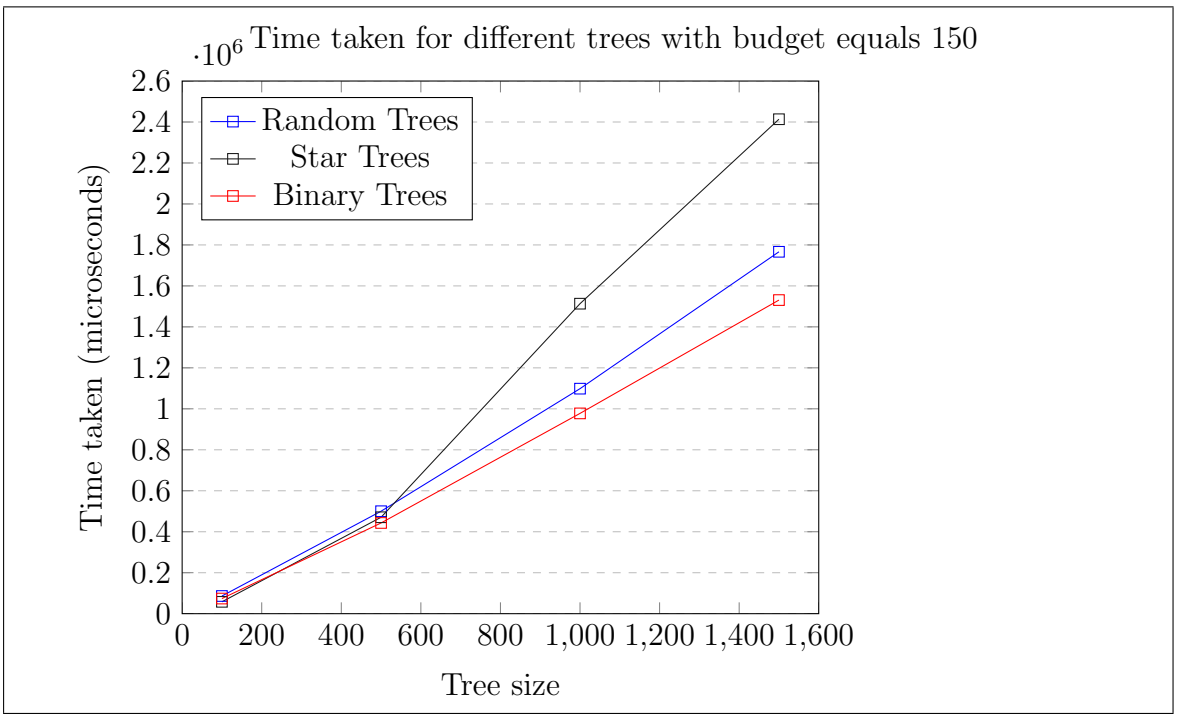


Figure 4.17: Different trees: running times for budget equals 150.

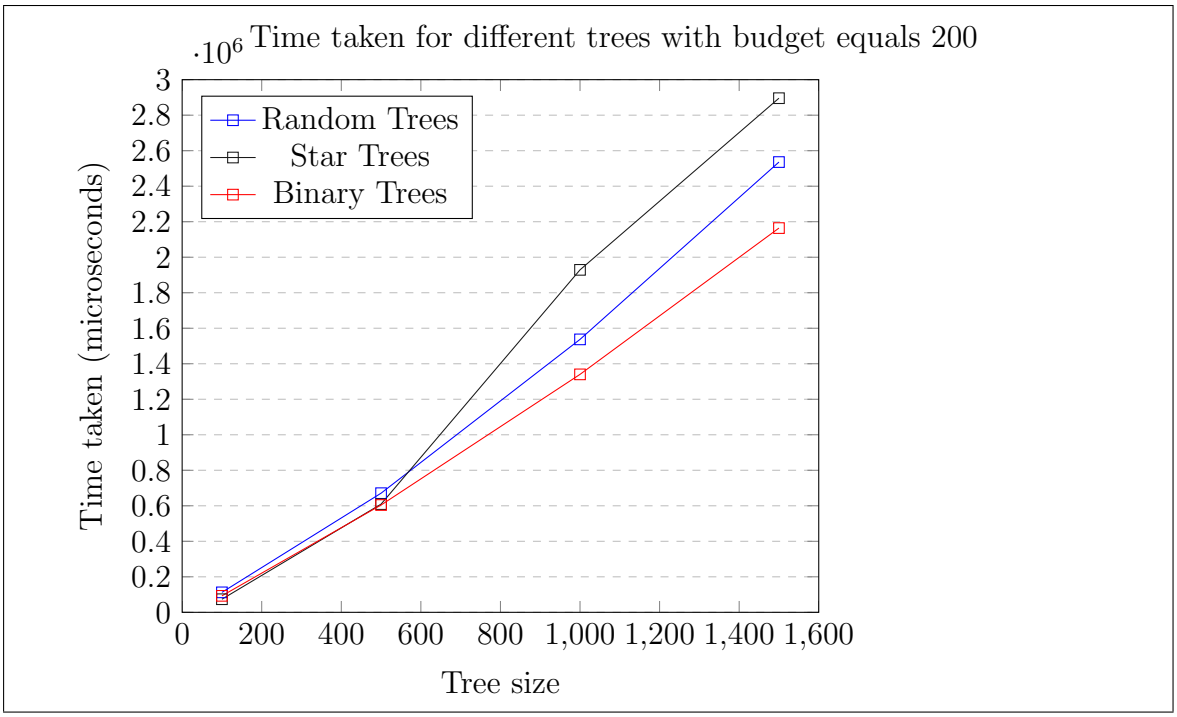


Figure 4.18: Different trees: running times for budget equals 200.

Tree Size	**Budget**	**Time Taken (microseconds)**	**Space used**
100	20	22992	7856
	50	33960	9480
	100	54906	13440
	150	73868	17580
	200	92822	22048
500	20	95792	10564
	50	165624	24168
	100	304254	62296
	150	442814	111796
	200	604415	172428
1000	20	186554	13956
	50	324620	42456
	100	623188	132528
	150	977656	259084
	200	1340194	411308
1500	20	281242	17284
	50	499817	60148
	100	967500	199676
	150	1530825	411756
	200	2164143	672012

Table 4.6: Binary tree running-time table 2

Chapter 5

Conclusion

In this chapter, we summarize the work done in this thesis and give some future work to be considered.

5.1 Contribution

In this thesis, we empirically analyze the dynamic programming algorithm in [1] for the budgeted maximum vertex cover problem on weighted trees (BMVCT). We defined the BMVCT problem and discussed different techniques for solving it. We identified the reasons for choosing the dynamic programming approach.

The dynamic programming algorithm was implemented with $C++$ programming language and compiled on *GNU GCC* compiler. Our implementation worked for all four variations of weighted trees. Trees with unweighted vertices and unweighted edges, trees with weighted vertices and unweighted edges, trees with unweighted vertices and weighted edges, and trees with weighted vertices and weighted edges.

We experimented the implemented code on random trees, binary trees and star trees. We used different tree sizes between 5 and 1500. We also used different budget values between 1 and 200. We measured the time taken and space used by the implemented code for the different trees. Binary trees used the least memory space for larger sized trees. Random trees were in between. Star trees used the largest space. On the average, binary trees performed best on our implementation considering their

running times and memory used.

The running times of the implemented code on our tests were all below three seconds. The lowest running time was 0 microsecond, the highest running time was about 2.89 seconds. On the average, the code used up considerable memory. The smallest memory space used was about $7.5MB$, the greatest around $3GB$ used by a star tree with 1500 vertices.

5.2 Future Work

We are interested in investigating how the algorithm's memory usage can be reduced. To do so, we have to study all the stored DP states. In particular, we want to find if there are states that are not used after a period that we can reuse to save memory space used.

Another interesting area we would like to extend this implementation to is the weighted partial vertex cover problem (WPVCT). WPVCT is similar to BMVCT except that it takes in the number of edges to be covered while BMVCT takes in a budget. In Mkrtchyan et al. [1], they explain how to convert the BMVCT algorithm into a subroutine for WPVCT.

We would also like to extend this implementation to graphs and other structures. In order to do this, we have to take cognizance of cycles and how they could impact the decisions in our dynamic programming table.

References

[1] Vahan Mkrtchyan, Ojas Parekh, Danny Segev, and K. Subramani, "The approximability of partial vertex covers in trees," in *SOFSEM 2017: Theory and Practice of Computer Science*, Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, Eds., Cham, 2017, pp. 350–360, Springer International Publishing.

[2] Michael Garey and David Johnson, *Computers and Intractability. A Guide To the Theory of NP-Completeness*, W. H. Freeman, Oxford, UK, 1979.

[3] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds., pp. 85–103. Plenum Press, 1972.

[4] M. R. Garey and D. S. Johnson, ""strong" np-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, July 1978.

[5] Paul E. Black, Algorithms, and Theory of Computation Handbook, ""tree", in dictionary of algorithms and data structures [online]," https://www.nist.gov/dads/HTML/tree.html, 2017, Accessed: 2019-11-11.

[6] Hung Q. Ngo, "Introduction to approximation algorithms," February 2005, URL: https://pdfs.semanticscholar.org/f4af/b4dc280f99fa00f90aa2d22bb3edf6d8300e.pdf.

[7] Williamson, David P, and David Bernard Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[8] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser, "Exceptional paper—location of bank accounts to optimize float: An analytic study of exact and approximate algorithms," *Management Science*, vol. 23, no. 8, pp. 789–810, 1977.

[9] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman, "Implementing data cubes efficiently," *SIGMOD Rec.*, vol. 25, no. 2, pp. 205–216, June 1996.

[10] Andreas Krause, *Optimizing Sensing: Theory and Applications*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2008, AAI3343461.

[11] David Kempe, Jon Kleinberg, and Éva Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2003, KDD '03, pp. 137–146, ACM.

[12] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims, "Learning diverse rankings with multi-armed bandits," in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, ICML '08, pp. 784–791, ACM.

[13] Yisong Yue and Thorsten Joachims, "Predicting diverse subsets using structural svms," in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, ICML '08, pp. 1224–1231, ACM.

[14] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Hooyeon Lee, "Approximating low-dimensional coverage problems," 2011.

[15] D Subhadrabandhu, Farhan Anjum, Sampath Kannan, and Sishir Sarkar, "Domination and coverage guarantees through distributed computation," 01 2005.

[16] Dorit S. Hochbaum and Anu Pathria, "Analysis of the greedy approach in problems of maximum k-coverage," *Naval Research Logistics (NRL)*, vol. 45, no. 6, pp. 615–627, 1998.

[17] Richard Loulou, "Minimal cut cover of a graph with an application to the testing of electronic boards," *Operations Research Letters*, vol. 12, no. 5, pp. 301 – 305, 1992.

[18] Oded Berman, Richard C Larson, and Nikoletta Fouska, "Optimal location of discretionary service facilities," *Transportation Science*, vol. 26, 08 1992.

[19] Oded Berman, Dimitris Bertsimas, and Richard C. Larson, "Locating discretionary service facilities, ii: Maximizing market size, minimizing inconvenience," *Operations Research*, vol. 43, no. 4, pp. 623–632, 1995.

[20] Barna Saha and Lise Getoor, *On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch*, pp. 697–708, 2009.

[21] Mitchell Jones, "The maximum facility location," M.S. thesis, University of Sydney, 2015.

[22] Reza Zanjirani Farahani, Nasrin Asgari, Nooshin Heidari, Mahtab Hosseininia, and Mark Goh, "Covering problems in facility location: A review," *Computers and Industrial Engineering*, vol. 62, no. 1, pp. 368 – 407, 2012.

[23] Alexander A. Ageev and Maxim I. Sviridenko, "Approximation algorithms for maximum coverage and max cut with given sizes of parts," in *Integer Programming and Combinatorial Optimization*, Gérard Cornuéjols, Rainer E. Burkard, and Gerhard J. Woeginger, Eds., Berlin, Heidelberg, 1999, pp. 17–30, Springer Berlin Heidelberg.

[24] Samir Khuller, Anna Moss, and Joseph Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, pp. 39–45, 1999.

[25] Nicola Apollonio and Bruno Simeone, "The maximum vertex coverage problem on bipartite graphs," *Discrete Applied Mathematics*, vol. 165, pp. 37 – 48, 2014, 10th Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2011).

[26] Bugra Caskurlu, Vahan Mkrtchyan, Ojas Parekh, and K. Subramani, "On partial vertex cover and budgeted maximum coverage problems in bipartite graphs," in *Theoretical Computer Science*, Josep Diaz, Ivan Lanese, and Davide Sangiorgi, Eds., Berlin, Heidelberg, 2014, pp. 13–26, Springer Berlin Heidelberg.

[27] Gwenaël Joret and Adrian Vetta, "Reducing the rank of a matroid," 2012.

[28] Nicola Apollonio and Bruno Simeone, "Improved approximation of maximum vertex coverage problem on bipartite graphs," *SIAM Journal on Discrete Mathematics*, vol. 28, 07 2014.

[29] Reuven Cohen and Liran Katzir, "The generalized maximum coverage problem," *Information Processing Letters*, vol. 108, no. 1, pp. 15 – 22, 2008.

[30] B. Kar, E. H. Wu, and Y. Lin, "The budgeted maximum coverage problem in partially deployed software defined networks," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 394–406, Sep. 2016.

[31] Kaiyue Zheng, Laura A. Albert, James R. Luedtke, and Eli Towle, "A budgeted maximum multiple coverage model for cybersecurity planning and management," *IISE Transactions*, vol. 51, no. 12, pp. 1303–1317, 2019.

[32] C. Fragoudakis E. Markou, S. Zachos, "Budgeted coverage of a maximum part of a polygonal area," 2003, pp. 174–182, 1st Balkan Conference on Informatics (BCI' 03) Thessaloniki, Greece.

[33] Sheng Zhang, Xingquan Wang, and Shanglu He, "Improved greedy algorithm for maximum coverage problem with group budget constraints," *International Journal of Pure and Applied Mathematics*, vol. 41, no. 6, pp. 755 – 759, 2007.

[34] Dror Rawitz and Adi Rosén, "Online budgeted maximum coverage," in *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[35] Donald E Curtis, Sriram V Pemmaraju, and Philip Polgreen, "Budgeted maximum coverage with overlapping costs: monitoring the emerging infections network," in *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2010, pp. 112–123.

[36] S. Wang, W. Zhao, and C. Wang, "Budgeted cell planning for cellular networks with small cells," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4797–4806, Oct 2015.

[37] Pasin Manurangsi, "A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation," 2018.

[38] Vangelis Th. Paschos, "A polynomial time approximation schema for maximum k-vertex cover in bipartite graphs," 2019, URL: https://arxiv.org/abs/1909.08435.

[39] Bugra Caskurlu, Vahan V. Mkrtchyan, Ojas Parekh, and K. Subramani, "Partial vertex cover and budgeted maximum coverage in bipartite graphs," *SIAM J. Discrete Math.*, vol. 31, pp. 2172–2184, 2017.