Team Control Number

10751

Problem Chosen

B

2020
HiMCM
Summary Sheet

Helping Florida to use the funds in an effective and balanced way is a problem related to the optimization with multiple constraints. To solve the difficult problem, we developed a knapsack model and a scheduling model to analyze the properties of the projects involved in the conservation plan and find out what mathematical process we are going to go through to reach our goal. We also modified the algorithm for knapsack problem to solve the model and derive the result: the best schedule of the plan.

We have given formal analysis over all the concepts related to our model, including the raw attributes of the projects given in the attachments of the problem and the derived attributes of the plan and projects calculated from the raw attributes. After analyzing the factors we need to consider for the problem, we find that the problem can be reduced into two smaller problems: a knapsack problem and a scheduling problem.

The knapsack problem is to select proper projects so that the efficiency of the plan is maximized. We have used a modified algorithm for knapsack problem to solve the knapsack model. After this step, we have derived the best choice of projects.

The scheduling problem is to decide the start time for the projects in the plan so that the funds spent over time is most balanced. We have used mathematics to reduce the problem into a minimization of sum of square, and used a rather brute force algorithm to get the result. This gives us the final schedule of the conservation plan.

Finally, we reviewed our model and listed its pros and cons. Although some of the general assumptions of the model make its application limited, the comprehensiveness and effectiveness of our model is significant.

Contents

1 Introduction

It is a global issue that the decrease in biodiversity due to the extinction of endangered plants is a serious problem for the balance of the ecosystem. To avoid this, people need to spend money on protecting the plants and animals.

However, in some places like Florida, greatly needing biodiversity conservation, people do not have enough money to protect all of the imperiled species there. They need to find out a proper scheme of funding and protecting to make full use of the funds to get as much benefit as possible.

Different projects of conservation have different cost, benefit, and required time, as shown in the table of threatened plants data (abbreviated as TPD in the following parts of the paper). It is to be determined which projects are selected in the optimal plan according to these factors. After choosing the projects, when to start the projects should also be determined to balance the funds spent over time as possible.

To make it clear what is going to be done, the problem is restated in Section 2. The model used in our solution to the problem and its explanation are shown in Section 5. The algorithms used to solve the model are introduced and explained in Section 6. We will also write a non-technical memo in Section 8 to give our proposal according to the result of our model.

2 Restatement and analysis of the problem

Various plant conservation projects with different budgets and timelines are to be conducted. However, one major issue is that funds are not sufficient to support all projects. Therefore, it is essential to make a plan for the managers who monitor the whole plan to help them precisely decide which project should be included and when and how to allocate their fund.

What is notable is that the final goal of this plan is not to simply save as many as plant species as possible. Instead, it should generate greatest benefit meanwhile taking account of different factors, like feasibility of success and the uniqueness of this species. The time spanning to finish the whole plan is also a factor that needs to be seriously considered because as more time being taken to protect one species, the likelihood of another plant dying out shall increase, finally leading to less benefit. Another factor is the funds spent on these projects. Obviously, a project with median benefit but high expense does not fit. The most suitable plan we need to find out should include all variables mentioned and give the best result.

The mathematical description of the problem is given in Section 5. To be specific, the problem is to reach the goal described in Section 5.8.

The factors that we need to consider are

- The total benefit of the plan should be larger as possible;
- The time spent on the plan should be shorter as possible;
- Considering the taxonomic uniqueness of protected species, the benefit of a project may not be fully exerted;
- Considering the feasibility of success, a project may not be successfully executed;
- The funds spent over time should be balanced as possible.

These factors are coupled in different ways and have different priorities. Therefore, our work is going to consist of

1. Giving the mathematical definition and explaining the factors above and related concepts,
2. Analyzing the factors and deriving the way that they are coupled together,

3. Determining the priorities of the factors and thus determining the goals of our mathematical model,

4. Developing algorithms to solve the model and thus reaching the determined goals.

The whole process is mainly shown in Section 5, explaining the mathematical model, and in Section 6, explaining the algorithms.

3 List of symbols

To make our model concise and straight forward, a list of symbols are defined as following, as shown in Table 1. The meaning of the symbols are to be explained in detail in the rest of this article. In the table, the symbols in lower case are variables w.r.t. a project, and the symbols in upper case are variables w.r.t. the plan. The difference between a project and the plan is described in Section 5.1.

Note that in this article, all numberings (like x here numbering the projects, and n later numbering the years) start with 0 instead of 1 to meet the convention in computer science.

The symbol $\sum_j^n$ means to sum for $j = 0, 1, \ldots, n-1$. Similar notations are used for other operators like $\prod$, Var (variance), and so on.

If X is a variable related to the plan, X^* denotes the actual value of the variable in the final optimal plan.

Table 1: List of symbols

Symbol	Domain	Unit	Meaning
x	$\mathbb{N}$		The index of projects
D	$\mathbb{N}$	year	Duration of the plan
d_x	$\mathbb{N}$	year	Duration of project x
n	$\mathbb{N}$	year	Time (index of years)
Z	$\mathscr{P}(\mathbb{N})$		The set of chosen projects
F	$[0, +\infty)$	dollar	Total funds
b_x	$[0, +\infty)$	osu[1]	Benefit of project x
u_x	$[0, 1]$		Taxonomic uniqueness of project x
s_x	$[0, 1]$		Feasibility of success of project x
$c_{x,n}$	$[0, +\infty)$	dollar	Cost of project x in the nth year
c_x	$[0, +\infty)$	dollar/year	Total cost of project x
C_n	$[0, +\infty)$	dollar/year	Cost of the plan in the nth year
C	$[0, +\infty)$	dollar	Cost of the plan
β_x	$[0, +\infty)$	osu	Effective benefit of project x
B	$[0, +\infty)$	osu	Total effective benefit of the plan
T	$[0, +\infty)$	osu/year	Time efficiency of the plan
ξ	$\mathbb{N}$		The longest project in the plan
a_x	$\mathbb{N}$	year	When project x starts

4 General assumptions

Assumption 1. *The funds F are provided off at one time.*

[1]The unit osu is invented to represent the unit of benefit.

We do consider the situation that different funds are raised in different years, but when adjusting a_x in the plan, not when deciding $\boldsymbol{Z}$. This can simplify the constraint given by the limitation of funds.

If we consider that the funds may be given differently in different years at the first place as a constraint for our plan, Equation 10 will be written in a far more complicated way, which invalidates all the algorithms and theorems stated in Section 6. In fact, without this assumption, the problem becomes NP-hard because it can be reduced to a multi-dimensional knapsack problem [1].

This assumption together with Assumption 5 gives the priority of optimization.

Assumption 2. *A project cannot be paused.*

In this way, the funds spent each year can only be adjusted by adjusting the start time for each project, but cannot be adjusted by pausing some certain projects. In other words, it is impossible that the plan requires to pause a project because funds are used up. This makes it possible that the funds spent by a project during a plan can be described according to only a single number a_x.

This assumption also avoids the possibility that a project fails midway when executed, creating better choices for the rest of the plan for remediation.

Assumption 3. *Different projects can be executed simultaneously and do not affect each other.*

We do not consider that different projects may have some actions in common and share their cost, or one project benefit another project and make more total benefit. Such situations can make the problem extremely hard because the benefits and costs of the projects are coupled in a nonlinear way instead of simplify summed up.

This assumption is used when defining $\boldsymbol{B}$ in Section 5.3 because it can only be defined as Equation 2 when there are no interrelationships between projects. It is also used when defining $\boldsymbol{C}_n$ in Section 5.6 because it can only be defined as Equation 7 if there are no interrelationships between projects.

Assumption 4. *The plan should not change when it is ongoing.*

This assumption is necessary to make our result a simple schedule instead of a complicated flow chart.

Assumption 5. *The maximization of $\boldsymbol{T}$ is prior to the balancing of funds spent over time (minimizing $\mathrm{Var}_n^D \boldsymbol{C}_n$).*

This assumption together with Assumption 1 plays an important role when describing our final goal of the model in Section 5.8.

5 The model

5.1 Projects and the plan

The word **project** is used to represent a conservation of a certain species of plant. According to the TPD, there are some **raw data** attributed to a project x, which are the **unique id**, the **taxonomic uniqueness** u_x, the **feasibility of success** s_x, the **cost** $c_{x,n}$ of the nth year. These concepts are explained in Section 5.2.

The word **plan** is used to represent how the projects are going to be executed, including which projects are executed (the set $\boldsymbol{Z}$) and when they should start (the sequence $\{a_x\}$). Deciding the which projects to be executed is to maximize the time efficiency (described in Section 5.5). Deciding when the projects start is to minimize the variance of yearly cost (described in Section 5.6).

5.2 Raw attributes of projects

To make the concepts more clear and make it easy to use them later, although the meanings of the raw data are described in the TPD, their meanings should be re-described in a mathematical way.

Definition 1 (benefit). *The **benefit** of project x is a real number $b_x \in [0, +\infty)$.*

The benefit is an important indicator indicating how imperiled the species is and how easy the project can be done. However, b_x cannot be directly used to measure how much can we benefit from finishing the project. Another derived concept called **effective benefit** should be used instead, as explained in Section 5.3.

Definition 2 (taxonomic uniqueness). *The **taxonomic uniqueness** of project x is a real number $u_x \in [0, 1]$.*

The taxonomic uniqueness is a measure of how the species is unique from other species. It is stipulated that $u_x = 0$ if there exists a species that is exactly the same as the species conserved in project x, and $u_x = 1$ if the species conserved in project x does not have any similarities with any other species.

Definition 3 (feasibility of success). *The **feasibility of success** of project x is a real number $s_x \in [0, 1]$.*

The meaning of s_x is the probability of the success of project x.

Definition 4 (yearly cost). *The **yearly cost** of project x is a real number $c_{x,n} \in [0, +\infty)$.*

The meaning of $c_{x,n}$ is cost of project x in the nth year.

5.3 Effective benefit

Although b_x is defined for a project, hardly can a project lead to so much benefit finally. A new concept should be defined to describe how much can people actually benefit from finishing the project.

Definition 5 (effective benefit). *The **effective benefit** of a project is a real number $\beta_x \in [0, +\infty)$ defined as*

$$\beta_x := b_x u_x s_x. \tag{1}$$

Here is the explanation of why the effective benefit should be defined as Equation 1.

First, consider the effect of u_x. If $u_x = 0$, which means there is a species the same as the conserved species, the conservation is useless because even if the conserved species die out, there are other species taking the place. If $u_x = 1$, which means there are no similar species to the conserved species, the conservation is meaningful and can exert full benefit. From these two cases, it is reasonable to multiply b_x with u_x, which means u_x represents the portion of b_x that can be exerted if the project is successfully finished.

Then, consider the effect of s_x. It is a probability, so the effect should be considered statistically. Imagine there are many parallel universes, in each of which project x is carried out under the same condition. According to the law of large numbers, the portion of successful executions is the probability s_x, exerting the benefit, while the rest does not make any benefit. Then, the mean of all the benefits is $s_x \cdot b_x + (1 - s_x) \cdot 0 = b_x s_x$.

Combining the two effects above, the formula for effective benefit (Equation 1) can be derived.

From now on, when measuring how much can a project benefit people, β_x is used instead of b_x.

By summing up the effective benefit for all projects of the plan, the total effective benefit can be defined.

Definition 6 (total effective benefit)**.** *The* ***total effective benefit*** *of the project is a real number* $\boldsymbol{B} \in [0, +\infty)$ *defined as*

$$\boldsymbol{B} := \sum_{x \in Z} b_x. \tag{2}$$

The $\boldsymbol{B}$ is an important indicator of how much a plan is expected to benefit if the plan is executed.

5.4 Duration

The **duration** of a project is the time for which it lasts. It can be defined according to the yearly costs of the project.

Definition 7 (duration of project)**.** *The* ***duration of project*** x *is a real number* $d_x \in \mathbb{N}$ *defined as*

$$d_x := \min \left\{ m \in \mathbb{N} \middle| \forall n \geq m : c_{x,n} = 0 \right\} \tag{3}$$

According to the TPD, a project must be able to end, which means such d_x must exist.

The duration of the plan should be defined as the time interval between the time when the first project of the plan starts and the time when the last project of the plan ends. However, due to Assumption 5, the duration of the plan should unnecessarily be defined in a so straight-forward way. A better definition is given in Equation 6 in Section 5.5.

5.5 Time efficiency

Definition 8 (time efficiency)**.** *The* ***time efficiency*** *of the plan is a real number* $\boldsymbol{T} \in [0, +\infty)$ *defined as*

$$\boldsymbol{T} := \frac{\boldsymbol{B}}{\boldsymbol{D}}. \tag{4}$$

The time efficiency of the plan is the average benefit gain during a unit of time. It should be maximized to make the plan as efficient as possible, whose reason can be found in Section 5.8.

According to Assumption 5, $\boldsymbol{T}$ should be maximized before we think about how a_x can be adjusted to balance the funds spent over time. To maximize the time efficiency, the duration of the plan is the same as the duration of the project with the largest duration (ξ) in the plan, where ξ is the project with largest duration among $\boldsymbol{Z}$

$$\xi := \arg\max_{x \in Z} d_x. \tag{5}$$

Then, when a_x are adjusted to minimize variance of yearly costs, the start time of projects with shorter duration than d_ξ will never be changed to make them start before ξ starts or end after ξ ends because otherwise $\boldsymbol{D}$ increases and thus makes $\boldsymbol{T}$ smaller. Therefore, rather than the straight-forward definition shown in Section 5.4, there is a better definition.

Definition 9 (duration of the plan)**.** *The* ***duration of the plan*** *is a real number* $\boldsymbol{D} \in \mathbb{N}$ *defined as*

$$\boldsymbol{D} := \max_{x \in Z} d_x. \tag{6}$$

It is obvious that if $\boldsymbol{C}$ stays the same, the larger $\boldsymbol{T}$ is, the better the plan is. In fact, maximizing $\boldsymbol{T}$ is of the most priority. It may be surprising that the most prior goal is not maximizing $\frac{\boldsymbol{T}}{\boldsymbol{C}}$, which is more intuitive. We do not try to maximize $\frac{\boldsymbol{T}}{\boldsymbol{C}}$ because if we did that, it would lead to a trivial result: the plan only contains one project with the minimum $\frac{\beta_x}{d_x c_x}$, which is not what we want to see.

Proof. Without loss of generality, assume that there are only 2 plant candidates for the plan, and then the mathematical induction can be used to generalize the conclusion to any number of plant candidates.

Without loss of generality, assume that $d_0 \le d_1$, which means $D = d_1 \ge d_0$.

Case 1: $\frac{\beta_0}{c_0} \le \frac{\beta_1}{c_1}$.

In this case,

$$\frac{\beta_0}{c_0} \le \frac{\beta_0 + \beta_1}{c_0 + c_1} \le \frac{\beta_1}{c_1}.$$

Thus, if the plan consists of both the 2 projects,

$$\frac{T}{C} = \frac{\beta_0 + \beta_1}{c_0 + c_1} \cdot \frac{1}{d_1} \le \frac{\beta_1}{c_1 d_1}.$$

Therefore, the plan consisting of only project 1 is better than that of both projects.

Case 2: $\frac{\beta_0}{c_0} > \frac{\beta_1}{c_1}$.

Similarly to case 1, in this case,

$$\frac{\beta_0}{c_0} > \frac{\beta_0 + \beta_1}{c_0 + c_1} > \frac{\beta_1}{c_1}.$$

Thus, if the plan consists of both the 2 projects,

$$\frac{T}{C} = \frac{\beta_0 + \beta_1}{c_0 + c_1} \cdot \frac{1}{d_1} < \frac{\beta_0}{c_0 d_1} \le \frac{\beta_0}{c_0 d_0}.$$

Therefore, the plan consisting of only project 0 is better than that of both projects.

From the 2 cases above, it can be derived that there always exists a plan with single project having the largest $\frac{T}{C}$ compared to any other plan. □

5.6 Variance of yearly costs

First, yearly costs need defining. The yearly cost in the nth year of the plan is the sum of the costs of the projects in the nth year. It should take into account the different start time of the projects.

Definition 10 (start time)**.** *The* ***start time*** *of project x is a real number $a_x \in \mathbb{N}$.*

Definition 11 (yearly costs)**.** *The* ***yearly cost*** *of the plan in the nth year is a real number $C_n \in [0, +\infty)$ defined as*

$$C_n := \sum_{x \in Z} c_{x,n-a_x}. \tag{7}$$

To measure how balanced the spent funds are distributed through time, the variance of yearly costs $\mathrm{Var}_n^D C_n$ is defined. In Section 5.8, it is going to be minimized to balance the funds spent over time.

It can be proved that minimizing $\mathrm{Var}_n^D C_n$ is the same as minimizing $\sum_n^D C_n^2$.

Proof. In this proof, symbol "$\sim$" denotes that two expressions have the same monotonicity.

$$\begin{aligned}
\operatorname*{Var}_n^D C_n &= \frac{1}{D}\sum_n^D \left(C_n - \frac{C}{D}\right)^2 \\
&\sim \sum_n^D \left(C_n^2 - \frac{2C}{D}C_n + \frac{C^2}{D^2}\right) \\
&= \sum_n^D C_n^2 - \frac{2C}{D}\sum_n^D C_n + \sum_n^D \frac{C^2}{D^2} \\
&= \sum_n^D C_n^2 - \frac{C^2}{D} \\
&\sim \sum_n^D C_n^2.
\end{aligned}$$

Thus, minimizing $\operatorname{Var}_n^D C_n$ is the same as minimizing $\sum_n^D C_n^2$. □

5.7 Constraints to the plan

First, consider the constraint due to the limitation of funds. To describe the constraint, the total funds and the cost of the plan needs defining.

Definition 12 (total funds)**.** *The* ***total funds*** *is a real number* $F \in [0, +\infty)$*.*

Definition 13 (cost of project)**.** *The* ***cost of project*** *is a real number* $c_x \in [0, +\infty)$ *defined as*

$$c_x := \sum_n^\infty c_{x,n}. \tag{8}$$

The sum c_x must converge because all terms following the d_xth term are 0.

Definition 14 (cost of the plan)**.** *The* ***cost of the plan*** *is a real number* $C \in [0, +\infty)$ *defined as*

$$C := \sum_{x \in Z} c_x. \tag{9}$$

Due to Assumption 1, the constraint should be

$$C \le F. \tag{10}$$

There are also constraints resulting from the restrictions for a_x due to the optimization of T. According to the discussions in Section 5.5, any projects in the plan should not start before the longest project start or end after the longest project end. Let the start time of the longest project in the plan be the 0th year, and then we have constraints for the start time.

$$\forall x \in Z : a_x \in \left[0, D - d_x\right). \tag{11}$$

5.8 The goal of the model

The goal of the model is to

1. Decide proper Z to make the plan have the maximum T,
2. Decide the minimum F to satisfy step 1 under the constraint Equation 10,

3. Decide the a_x for each chosen project in step 1 to minimize $\mathrm{Var}_n^D C_n$ under the constraint Equation 11.

In this way, the derived plan can meet the requirements described in Section 2.

To reach the goal, we need to develop some approaches based on the model, which are described in Section 6.

6 The algorithms

In Section 5.8, it is stated that the first step is to maximize T regardless of constraint Equation 10. However, because it is actually not known whether max T can increase constantly as long as allowable C increase until all projects available is chosen in Z. To study the trend of how max T under constraint Equation 10 changes when F increases, the actual step of our approach is to

1. Develop a program to maximize B given F,
2. Develop a program based on the last program to maximize T given F,
3. Ensure that T has a maximum value T^* whenever $F \geq F^*$, where F^* is to be determined,
4. Develop a program to minimize $\mathrm{Var}_n^D C_n$.

6.1 Maximize B given F under constraint Equation 10

This subproblem is a 0-1 knapsack problem, regarding

- F as the capacity of the knapsack,
- c_x as the weight of the xth item,
- β_x as the value of the xth item.

There is a classic dynamic programming (DP) algorithm to solve knapsack problems [2], shown in Algorithm 1.

Algorithm 1 Classic DP algorithm for 0-1 knapsack problem

```
Integerize F
for each project x do
    Integerize c_x
end for
function KNAPSACK(F)
    Initialize m as an array whose elements are all 0
    Initialize p as an array whose elements are all ∅
    for each project x do
        for w from F down to c_x do
            if m[w] < m[w − c_x] + β_x then
                m[w] ← m[w − c_x] + β_x
                p[w] ← p[w − c_x] ∪ x
            end if
        end for
    end for
    return m[F], p[F]
end function
```

However, this algorithm does not fit with our problem here because F and c_x should be natural numbers to appear as index of elements in arrays like m and p. The conflict can be resolved by rounding the values into integers, but due to the large capacity of the knapsack and the large weight of the items, the algorithm cannot give a result in acceptable time until the rounding makes the inaccuracy so significant that the result is not convincing at all.

Thus, a new algorithm should be developed. The main reason for Algorithm 1 to be slow is that it traverse through a whole continuous part of array m whenever it traverse a project x, while most of the elements in m are useless and are unnecessary to be updated. However, whether an element in m is useful cannot be determined using the DP algorithm. Inspired by this thought, the recursive algorithm can be adopted because it can make the program consider small w values first and determine which w values later on are useful. The new algorithm is shown in Algorithm 2.

Algorithm 2 Recursive algorithm for 0-1 knapsack problem

```
function RECURSIVE(F, x)
    if x exceeds the limit of projects then
        return 0, ∅
    else if c_x > F then
        return RECURSIVE(F, x + 1)
    else
        m_1, p_1 ← RECURSIVE(F, x + 1)
        m_2, p_2 ← RECURSIVE(F − c_x, x + 1)
        if m_1 < m_2 + β_x then
            return m_2 + β_x, p_2 ∪ x
        else
            return m_1, p_1
        end if
    end if
end function
function KNAPSACK(F)
    return RECURSIVE(F, 0)
end function
```

However, when there are many projects, the recursive algorithm even takes more time than the DP algorithm because there some (F, x) being calculated multiple times. Since this recursive function is a pure function, a cache can be used to store previously calculated result. Nonetheless, since F and c_x are floating numbers, the probability of cache hit is low. To increase the probability of cache hit, the numbers should be rounded. In our case, they are rounded to 100 dollars.

There can also be some other optimizations. The projects can be sorted according to their costs, so that the condition branch $c_x > F$ can be designed to be a short-circuit evaluation, immediately returning $0, \varnothing$ because the rest projects are even more expensive.

The optimized version is shown in Algorithm 3.

Algorithm 3 is implemented in Ruby as part of Appendix A.

6.2 Maximize T given F under constraint Equation 10

The plan given by maximizing B may not have maximum T. The only thing that can be ensured is that T cannot increase without reducing D. According to Equation 6, D is determined by the longest project. Thus, if we want to increase T, we need to remove the longest project in the selected plan, and find another plan that maximizes B without using the project that we have just removed. The process is cycled until no projects are chosen.

Algorithm 3 Recursive algorithm for 0-1 knapsack problem, optimized with cache and short-circuit evaluation

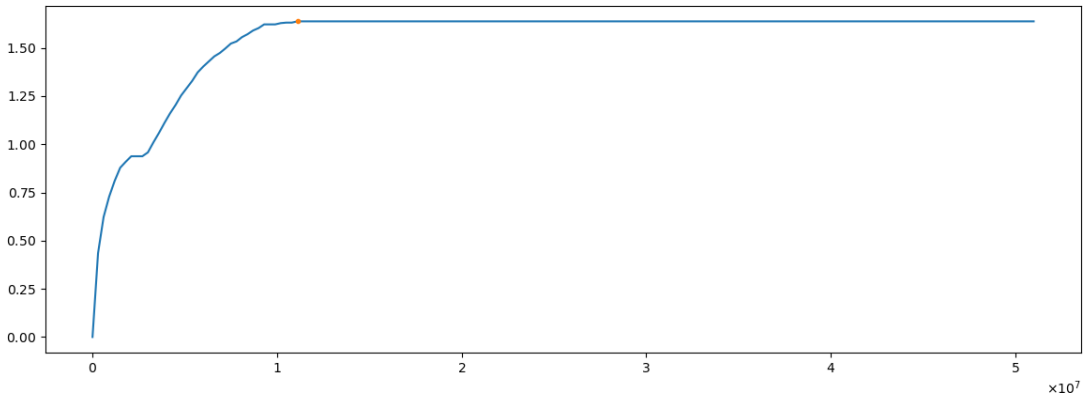
```
Integerize F
for each project x do
    Integerize c_x
end for
Sort the projects w.r.t. c_x
function RECURSIVE(F, x)
    if k[F, x] exists then
        return k[F, x]
    else if x exceeds the limit of projects or c_x > F then
        return k[F, x] ← 0, ∅
    else
        m_1, p_1 ← RECURSIVE(F, x + 1)
        m_2, p_2 ← RECURSIVE(F − c_x, x + 1)
        if m_1 < m_2 + β_x then
            return k[F, x] ← m_2 + β_x, p_2 ∪ x
        else
            return k[F, x] ← m_1, p_1
        end if
    end if
end function
function KNAPSACK(F)
    Initialize k as a key-value mapping
    return RECURSIVE(F, 0)
end function
```

From this insight, the pseudocodes shown in Algorithm 4. The algorithm is implemented in Ruby in Appendix A.

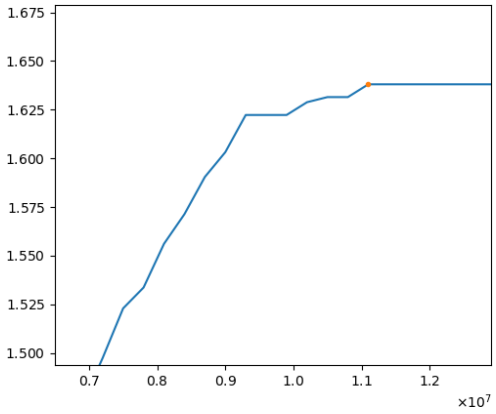
Algorithm 4 Algorithm for maximizing T given F

```
m, p ← 0, ∅
loop
    B, Z ← KNAPSACK(F)
    if Z = ∅ then
        return m, p
    end if
    if m < T then
        m, p ← T, Z
    end if
    Remove project ξ from the set of all projects
end loop
```

After running the program, the curve of max T related to F can be drawn, as shown in Figure 1. As can be seen, although F increases until there are enough funds to execute all the projects simultaneously, max T does not increase constantly but reaches its maximum when $F \geq F^*$. It can be derived that $F^* = 11\,010\,118.27$ dollars and $T^* = 1\,638\,017.60$ osu/year. The chosen plants are shown in Table 2 in Section 8.



(a) Overview of the whole graph, where the orange point is (F^*, T^*)



(b) Details of the graph near (F^*, T^*)

Figure 1: The graph of max T related to F

6.3 Minimizing $\mathrm{Var}_n^D C_n$ under constraint Equation 11

In Section 5.6, it has been proved that minimizing $\mathrm{Var}_n^D C_n$ is the same as minimizing $\sum_n^D C_n^2$, which can reduce some calculations in calculating the variance (while does not help come up with a smart algorithm because still, the problem is NP-hard).

Here, a brute force algorithm is adopted. It traverses through all possible start times (as many as $\prod_{x \in Z} (D - d_x)$ cases). The algorithm is implemented in Ruby in Appendix B.

After running the program, the optimal $\{a_x\}$ can be derived. The optimal $\{a_x\}$ are shown in Table 2 in Section 8 in form of a schedule. The minimum value of $\sum_n^D C_n^2$ is 26 457 147 921 183.15 dollars^2.

7 Pros and cons

Our model have some pros and cons in solving the problem described in 1. The pros are

- The model takes into account multiple factors. It considers the maximization of sum of effective benefit (taking into account the benefit, the taxonomic uniqueness, and the feasibility of success) of the projects in the plan, the minimization of the duration of executing the plan, and the balance of funds spent over time.
- The mechanism and theories about the model are simple and easy to understand. Although the algorithm may be difficult to be come up with, its mechanism and program

is simple, and its result can be reproduced easily.

- The model provides a method to output the actually chosen items in the recursive solution of a knapsack problem. It combines the advantages of the dynamic programming solution and the recursive solution: be fast regarding large knapsack problems and be able to output the contents of the optimal plan.

The cons are

- The model does not consider the funding schedule at first place. The funding schedule does not appear as a constraint as stated by Assumption 1.
- The model does not consider probable accidents during the plan. The plan cannot change midway even some projects stopped accidentally (which is not considered due to Assumption 2), creating better choices for the rest of the plan as remediation.
- The model does not consider the interrelationship between projects. It is normal that different projects include common actions which interrelates with each other, changing the cost. It is not considered as stated by Assumption 3.

8 The memo

Table 2 shows the schedule of the optimal plan. The *n*th column denotes the *n*th year. The last row sums the costs of the projects, representing the fundraising schedule.

The total cost of the plan is $11010118.27.

Table 2: Schedule of executing the projects

0	1	2	3	4
		1-Flowering Plants-502		
		1-Flowering Plants-436		
		1-Flowering Plants-536		
		1-Flowering Plants-183		
		1-Flowering Plants-480		
	1-Flowering Plants-135			
		1-Flowering Plants-481		
1-Flowering Plants-176				
		1-Flowering Plants-475		
1-Flowering Plants-546				
1-Flowering Plants-558				
1-Flowering Plants-553				
1-Flowering Plants-442				
		1-Flowering Plants-537		
		1-Flowering Plants-548		
1-Flowering Plants-426				
1-Flowering Plants-452				
		1-Flowering Plants-455		
		1-Flowering Plants-133		
1-Flowering Plants-168				
1-Flowering Plants-476				
		1-Flowering Plants-137		
1-Flowering Plants-485				
1-Flowering Plants-528				
1-Flowering Plants-520				
1-Flowering Plants-514				
1-Flowering Plants-517				
		1-Flowering Plants-529		
1-Flowering Plants-557				
1-Flowering Plants-179				
1-Flowering Plants-530				
1-Flowering Plants-440				
1-Flowering Plants-513				
1-Flowering Plants-524				
1-Flowering Plants-508				
1-Lichens-567				
$2600492.21	$2555863.21	$3024886.35	$1488407.44	$1340469.06

Appendices

A The program used to decide Z

```
require_relative 'plant'

BEGIN_TIME = Time.now
SORTED_TPD = TPD.sort

def puts_plants_set taken
  puts TPD.sum('') { taken.include?(_1) ? '|' : ' ' }
end

def max_effective_benefit_recursive remaining_funds, pos
  input = [remaining_funds, pos]
  return @cache[input] if @cache[input]
  return @cache[input] = [0, 0] if pos >= @plants.size || (plant =
   ↪ @plants[pos]).rounded_cost > remaining_funds
  benefit_if_not_take, set_if_not_take = max_effective_benefit_recursive
   ↪ remaining_funds, pos + 1
  benefit_if_take, set_if_take = max_effective_benefit_recursive
   ↪ remaining_funds - plant.rounded_cost, pos + 1
  benefit_if_take += plant.integerized_effective_benefit
  set_if_take = set_if_take.add plant
  @cache[input] = benefit_if_take > benefit_if_not_take ? [benefit_if_take,
   ↪ set_if_take] : [benefit_if_not_take, set_if_not_take]
end

def max_effective_benefit_under funds
  max_effective_benefit_recursive funds, 0
end

def max_time_efficiency_under funds
  @plants = SORTED_TPD.clone
  max_time_efficiency, max_time_efficiency_taken = 0r, 0
  loop do
    @cache = {}
    max_effective_benefit, taken = max_effective_benefit_under funds
    plant_to_be_removed, index_to_be_removed = @plants.each_with_index.max_by
     ↪ { |plant, i| taken.include?(plant) ? plant.duration : 0 }
    puts_plants_set taken
    break unless plant_to_be_removed && taken.include?(plant_to_be_removed)
    new_time_efficiency = max_effective_benefit.quo
     ↪ plant_to_be_removed.duration
    max_time_efficiency, max_time_efficiency_taken = new_time_efficiency,
     ↪ taken if new_time_efficiency > max_time_efficiency
    @plants.delete_at index_to_be_removed
  end
  [max_time_efficiency, max_time_efficiency_taken]
end

def max_efficiency_under
```

```
  funds = TPD.sum &:rounded_cost
  max_efficiency, max_efficiency_taken, max_efficiency_funds = 0r, 0, funds
  loop do
    puts "Funds: #{funds}"
    max_time_efficiency, taken = max_time_efficiency_under funds
    break if taken.empty?
    cost = taken.sum &:rounded_cost
    new_efficiency = max_time_efficiency / cost
    max_efficiency, max_efficiency_taken, max_efficiency_funds =
     ↪  new_efficiency, taken, cost if new_efficiency > max_efficiency
    funds = cost - 1
  end
  [max_efficiency, max_efficiency_taken, max_efficiency_funds]
end

(0..510000).step 3000 do |funds|
  IO.write 'output.txt',
           "#{funds}: #{max_time_efficiency_under(funds).join ' '}\n",
           mode: 'a'
end

puts "Time passed: #{Time.now - BEGIN_TIME}"
```

The `plant.rb` file defines the data structure of a plant and imports the constant `TPD`.

B The program used to decide $\{a_x\}$

```
require_relative 'plant'

BEGIN_TIME = Time.now

def Array.product array, *arrays
  array.to_a.to_enum :product, *arrays.map(&:to_a)
end

TAKEN = 14796153683959
TAKEN_ARRAY = TAKEN.to_a
DURATION = TAKEN.max_by(&:duration).duration

FACTOR = 100.0 / TAKEN.reduce(1) { _1 * (DURATION - _2.duration + 1) }

result = Array.product(*TAKEN.map { 0..DURATION - _1.duration
 ↪  }).min_by.with_index do |starts, i|
  print "\r#{(i * FACTOR).round}%"
  DURATION.times.sum do |year|
    TAKEN_ARRAY.zip(starts).sum do |plant, start|
      year >= start ? plant.costs[year - start] || 0 : 0
    end ** 2
  end
end

puts "\r100%"
```

```
p result
puts "Time passed: #{Time.now - BEGIN_TIME}s"
```

C The program used to plot the graph of max T related to F

```
from scanf import scanf
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()
funds_list = []
max_time_efficiency_list = []
with open('output.txt') as f:
  for line in f:
    funds, numerator, denominator, _ = scanf("%d: %d/%d %d", line)
    funds_list.append(funds * 1e2)
    max_time_efficiency_list.append(numerator / denominator * 1e-6)
ax.ticklabel_format(style='sci', scilimits=(-1, 1), useMathText=True)
ax.plot(funds_list, max_time_efficiency_list)
ax.plot(11100000, 8190088e-6/5, '.')
plt.show()
```

The program reads the result of the program in Appendix A.

D Table of max T related to F for $F < F^*$

If F is given, under the constraints given by Equation 10, T can be maximized by properly deciding Z. The max T related to F for $F < F^*$ is given in Table 3. For $F \geq F^*$, max T is the same as T^*. Data in Table 3 are calculated by the program in Appendix A.

Table 3: Table of max T related to F before T is maximized

F (dollars)	max T (osu/year)
0.00	0.00
300000.00	435567.00
600000.00	622541.67
900000.00	727932.67
1200000.00	810146.67
1500000.00	878511.67
1800000.00	909344.67
2100000.00	937911.67
2400000.00	937911.67
2700000.00	937911.67
3000000.00	957915.40
3300000.00	1010095.00
3600000.00	1058935.00
3900000.00	1111114.60
4200000.00	1160443.00
4500000.00	1203976.60
4800000.00	1253305.00

5100000.00	1291070.20
5400000.00	1328921.20
5700000.00	1373425.00
6000000.00	1403508.00
6300000.00	1430026.80
6600000.00	1456288.20
6900000.00	1473947.80
7200000.00	1497505.20
7500000.00	1522926.40
7800000.00	1533631.60
8100000.00	1556091.40
8400000.00	1571313.80
8700000.00	1590420.80
9000000.00	1603152.20
9300000.00	1622259.20
9600000.00	1622259.20
9900000.00	1622259.20
10200000.00	1628825.20
10500000.00	1631451.60
10800000.00	1631451.60
11100000.00	1638017.60

References

[1] Ariel Kulik and Hadas Shachnai. There is no eptas for two-dimensional knapsack. *Information Processing Letters*, 110(16):707 – 710, 2010.

[2] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):414–424, 1999.