Summary and implementation of “Project portfolio selection and scheduling optimization based on risk measure: a conditional value at risk approach”

Daniel Norouzi, Shahab Kiani

Introduction

A **project portfolio** is a group of projects that share and compete for the same resources of an organization. Project portfolios are considered **“powerful strategic weapons”** for implementing the intended corporate strategy.

Conventional Project Selection Techniques

Conventional project selection techniques assume the availability of accurate information and use financial metrics such as **discounted cash flow, net present value, return on investment and payback period** for evaluating projects. However, every project is a **unique endeavor** and thus obtaining accurate information at the initial project evaluation stage is difficult. This induces **uncertainty** in the estimation of **project cash inflows** resulting in financial **risk**.

Risk Management in Project Portfolios

The concept of **project portfolio management** is similar to that of **financial asset management**. Assets with individual risks and returns are grouped to form a profitable group. An approach of asset **diversification** is adopted to manage risks. Similarly, project portfolio diversification through selection of projects from different **categories**,

- **Breakthrough**
- **Platform**
- **Derivative**
- **R&D**

enables organizations to achieve competitive objectives of corporate strategy.

Numerous studies on financial asset management have applied the **risk** measure **conditional-value-at-risk (CVaR)** to achieve portfolios with a minimum risk of **severe low returns** and that yield **higher returns** compared with portfolios obtained through **maximization** of **expected returns**. The **advantage** of *CVaR* is that it considers the potential risk of severe low returns. However, **no study** in the project portfolio context has incorporated *CVaR* in objective function to obtain a **project portfolio** with least risk profile. The present study contributes by addressing this research gap.

A Case Study of a Dairy Firm

The current paper presents a case study of a dairy firm, which has identified **20** potential projects. The study captures the financial risk of the identified projects by using **normal** distribution for uncertain project **cash inflows**. In addition, it evaluates the **strategic alignment** scores and risk scores of **technical risk**, **schedule risk**, **economic and political risk**, **organizational risk**, **statutory clearance risk** of the projects by using an **Analytical Hierarchy Process (AHP)**. Further, it formulates three project portfolio selection and scheduling models namely, **risk-neutral (max_E)**, **risk-averse (max_CVaR)** and **combined compromise (max_E_CVaR)** models.

The **max_E** model maximizes the expected total net present value of the selected and scheduled project portfolios. The **max_CVaR** model minimizes (the risk of obtaining severe low total net present values) by maximizing $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ for a given confidence level α. The **max_E_CVaR** model seeks a compromise between the maximization of $E(\widetilde{TNPV})$ and $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$by combining them to form a suitable objective function.

Simulation Optimization

Simulation optimization is adopted as a solution methodology. Using the python library **Scipy**, the `scipy.optimize.minimize` optimizer with **Trust Region Optimization Method** `method='trust-constr'` is utilized for the implementation of simulation optimization. The results obtained using the **three models** are analyzed to generate insights for decision-making at **varying confidence levels**. A comparison of the results obtained using the models shows that the **max_CVaR** model ensures that the **lowest return** in the **worst** scenario is maximized to the greatest extent possible, thereby yielding high returns even when the confidence levels are low. This model exploits the **diversification** approach for risk management and its portfolios contain at **least one project** from **each** project category (derivative, platform and breakthrough). The results obtained using the **max_E_CVaR** model can be utilized by decision-makers to select and schedule project portfolios according to their **risk appetite** and **acceptable trade-off** between risk-averse and risk-neutral objectives.

Table of Contents

- [Results]

Case Study Characteristics

The upper management decided that the organisation should strategically focus on following:

- **New product (NP)**
- **Market expansion (ME)**
- **Efficiency improvement (EI)**
- **Infrastructural development (ID)**
- **Environmental and social responsibility (ES)**

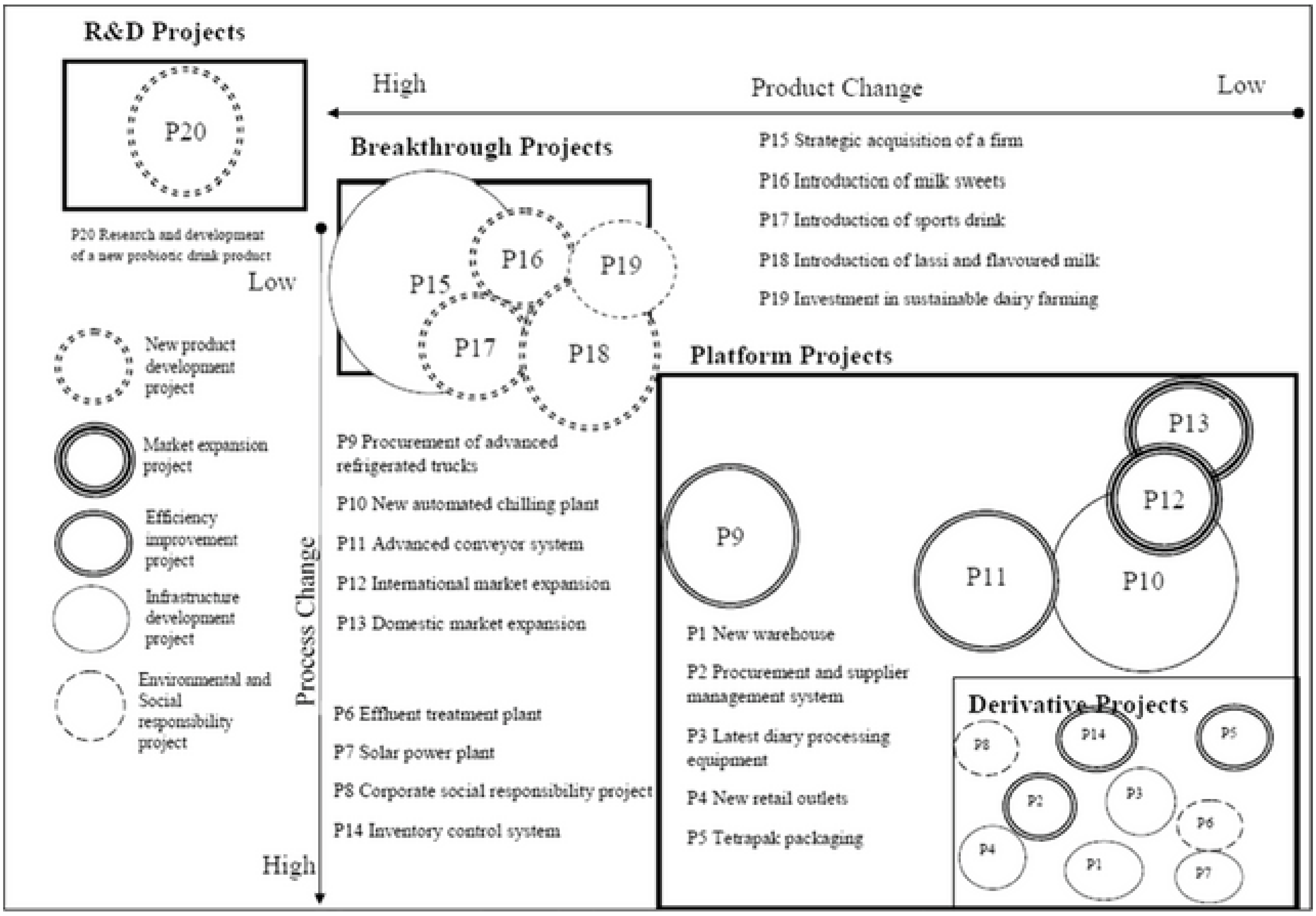
The **20** identified projects are diveresed in the following **categories**:

$$\text{Breakthrough} : \{P15, P16, P17, P18, P19\}$$

$$\text{Platform} : \{P9, P10, P11, P12, P13\}$$

$$\text{Derivative} : \{P1, P2, P3, P4, P5, P6, P7, P8, P14\}$$

$$\text{R\&D} : \{P20\}$$



Projects

The projects incur high **cash outflows** c_i, therefore it is crucial for the organization to select projects that align with the strategy, have minimum risk exposure, and yield maximum expected total net present value. Projects **P6** and **P8** are **mandatory** due to government regulations and social responsibility. The **total investment** required for the **20** identified projects is **USD 2000**. But the organization has total budget **availability** of only **USD 700 equally** spread over **four** stages (Inv_{s_1}, to Inv_{s_4} all equal to **USD 175**) each with **five** time units

$$S_1 = \{1, 2, 3, 4, 5\}; S_2 = \{6, 7, 8, 9, 10\}; S_3 = \{11, 12, 13, 14, 15\}; S_4 = \{16, 17, 18, 19, 20\}.$$

The weighted average **cost of capital** for the firm is estimated to be $wacc = 10\%$.
The characteristics of our diary farm case study is as follows:

Strategic Alignment Scores

- Project portfolios are formulated to achieve the intended corporate **strategy**, so it is crucial to ensure that the selected projects **align** with the organizational **goals**.
- The identified projects are evaluated to obtain their **"Strategic Alignment"** scores st_sc_i which are is critical for enhancing organizational performance and achieving long-term success.
- The projects in the **five strategic focus categories** are compared using **AHP** based on the **upper management's judgment.**
- **Table 1** bellow shows the pairwise **comparison matrix** and computed **strategic alignment scores**.
- The **highest** importance is given to **NP** development projects to attract more customers by offering diversified products.
- The **second highest importance** is given to **ME** projects to increase the market share and sales, followed by **EI** projects, **ID** and **ES** projects in that order.
- To **ensure** strategic alignment of the selected project portfolio, the upper management suggested that it should have the total strategic alignment score **greater than or equal** to $\text{st\_min} = 1$.
- **Note**: In the original paper the minimum total strategic alignment score was set to $\text{st\_min} = 2$ but the **provided** solutions **violated** the constraint so we reduced the parameter to $\text{st\_min} = 1$.

Table 1

	NP	ME	EI	ID	ES	st_sc_i
NP	1	3	5	7	9	0.503
ME	0.333	1	3	5	7	0.26

	NP	ME	EI	ID	ES	$\mathrm{st\_sc}_i$
EI	0.2	0.333	1	3	5	0.134
ID	0.143	0.2	0.333	1	3	0.068
ES	0.111	0.143	0.2	0.333	1	0.035

Uncertain Cash Inflows

- Every project is a **unique** endeavor, and thus, obtaining **accurate** information at the initial project evaluation stage is difficult.
- This induces **uncertainty** in the estimation of **cash inflows** of the project resulting in financial risk.
- Therefore, the present study captures the cash inflows of the projects by using **normal** distribution, such that the nth uncertain cash inflow from project i $\widetilde{\mathrm{n\_in\_cash}_i}$ has **mean** $\widetilde{\mathrm{n\_in\_cash}_i}(\mu)$ and **standard deviation** $\widetilde{\mathrm{n\_in\_cash}_i}(\sigma)$.

Risk Scores

- Because the projects operate in complex **uncertain** environment, they are also associated with different **risk types** namely, as shown in **Table 2**:
  - **Technical risk** ($\mathrm{Tech\_risk}$)
  - **Schedule risk** ($\mathrm{Sch\_risk}$)
  - **Economic & Political risk** ($\mathrm{EP\_risk}$)
  - **Organizational risk** ($\mathrm{Org\_risk}$)
  - **Statutory clearance risk** ($\mathrm{Stc\_risk}$).
- However, the **weightage** of each risk is different and the **severity** of risks varies across projects. Therefore, **weights** of the risks are calculated using the **AHP** based on the **upper management**'s judgment regarding their relative importance.
- **Table 3** presents the pairwise **comparison matrix** and computed **weights** of **technical risk** ($w_{\mathrm{Tech\_risk}}$), **schedule risk** ($w_{\mathrm{Sch\_risk}}$), **economic and political risk** ($w_{\mathrm{EP\_risk}}$), **organizational risk** ($w_{\mathrm{Org\_risk}}$) and **statutory clearance risk** ($w_{\mathrm{Stc\_risk}}$).
- The **highest** importance is given to **economic and political** risk followed by **technical** risk, **schedule** risk, **statutory clearance** risk and **organizational** risk in that order.
- Additionally, the upper management and risk experts of the domain **examined** each risk with respect to **each project** and assigned a **score** based on the **severity** of risk ($\mathrm{Tech\_risk}_i$; $\mathrm{Sch\_risk}_i$; $\mathrm{EP\_risk}_i$; $\mathrm{Org\_risk}_i$; $\mathrm{Stc\_risk}$).
- The **risk score** of a project is computed as the **weighted sum** of its individual risks.
- The **total risk score** of the portfolio is the **sum** of risk scores of all the selected projects.
- To ensure that the risk exposure of the firm is within the **acceptable** limit, the upper management suggested that the selected project portfolio should have a **total** weighted risk score **less than or equal** to $\mathrm{risk\_max} = 100$.

Table 2

Technical risk	**Schedule risk**	**Economic and political risk**	**Organizational risk**	**Statutory clearance risk**
Equipment risk	Project delay risk	Change in government policy risk	Supplier risk	Land acquisition risk
Technology selection risk	Improper estimates risk	Inflation risk	Contractor risk	Environmental clearance risk
Engineering and design change risk				

Table 3

	Tech_risk	**Sch_risk**	**EP_risk**	**Org_risk**	**Stc_risk**	**Weightage**
Tech_risk	1	3	0.33	7	5	0.260225
Sch_risk	0.33	1	0.2	5	3	0.13435
EP_risk	3	5	1	9	7	0.502825
Org_risk	0.143	0.2	0.11	1	0.33	0.03482
Stc_risk	0.2	0.33	0.143	3	1	0.06778

Table of Notations

Notation	**Description**
$T = \{1, 2, 3 \ldots 20\}$	Set of time units
$S_1 = \{1, 2, 3, 4, 5\}$	Set of time units for the first investment stage
$S_2 = \{6, 7, 8, 9, 10\}$	Set of time units for the second investment stage
$S_3 = \{11, 12, 13, 14, 15\}$	Set of time units for the third investment stage
$S_4 = \{16, 17, 18, 19, 20\}$	Set of time units for the fourth investment stage
t	Index for time unit
$I = \{1, 2, 3 \ldots .20\}$	Set of projects
i	Index for projects
Inv_{s_1}	Investment available for the first stage
Inv_{s_2}	Investment available for the second stage
Inv_{s_3}	Investment available for the third stage
Inv_{s_4}	Investment available for the fourth stage
o_{s_1}	Unutilized investment of the first stage overflowing to the second stage
o_{s_2}	Unutilized investment of the second stage overflowing to the third stage

Notation	Description
o_{s_3}	Unutilized investment of the third stage overflowing to the fourth stage
c_i	Initial investment required for project i
$n \in \{1, 2, 3, \ldots 10\}$	Index for cash inflow
$\widetilde{\text{n\_in\_cash}}_i$	nth uncertain cash inflow from project i
$\widetilde{\text{n\_in\_cash}}_i(\mu)$	Mean of normal distribution of nth cash inflow from project i
$\widetilde{\text{n\_in\_cash}}_i(\sigma)$	Standard deviation of normal distribution of nth cash inflow from project i
wacc	Weighted average cost of capital of the firm
st_sc_i	Strategic alignment score of project i
st_min	Minimum acceptable total strategic alignment score of portfolio
Tech_risk_i	Technical risk score of project i
$w_{\text{Tech\_risk}}$	Weightage assigned to the technical risk
Sch_risk_i	Schedule risk score of project i
$w_{\text{Sch\_risk}}$	Weightage assigned to the schedule risk
EP_risk_i	Economic and political risk score of project i
$w_{\text{EP\_risk}}$	Weightage assigned to the economic and political risk
Org_risk_i	Organizational risk score of project i
$w_{\text{Org\_risk}}$	Weightage assigned to the organizational risk
Stc_risk	Statutory clearance risk score of project i
$w_{\text{Stc\_risk}}$	Weightage assigned to the statutory clearance risk
risk_max	Maximum acceptable total risk score of portfolio
$\widetilde{NPV}_i$	Net present value of project i
$\widetilde{TNPV}$	Total net present value of project portfolio
$E(\widetilde{TNPV})$	Expected total net present value of project portfolio
α	Confidence level at which analysis is performed
$\text{VaR}_\alpha(\widetilde{TNPV})$	Value-at-risk of total net present value of project portfolio at confidence level α
$\text{CVaR}_\alpha(\widetilde{TNPV})$	Conditional-value-at-risk of total net present value of project portfolio at confidence level α
λ	Weightage of $E(\widetilde{TNPV})$ in the objective function of the composite compromise model
X_{it}	Decision variable such that $X_{it} = \begin{cases} 1 & \text{if project } i \text{ is launched at time } t \\ 0 & \text{otherwise} \end{cases}$

Profit & Risk Measures

We define the **profit** as the **total net present value** $TNPV$ of the project portfolio and measure the **risk** using **value-at-risk** (VaR) and **conditional value-at-risk** (CVaR). All these measures are calculated using the modules of the `TNPV` **python** class provided in the source code.

```
# Class to calculate TNPV, risk measures and check the constraints,
class TNPV:

    def __init__(
        self,
        projects, # Projects data
        risk_weights, # Risk weights for all risk types
        risk_max=100,
        st_min=0.2,
        wacc=0.1,
        times=np.array(range(1, 21)), # Portfolio timeline
        cash_inflow_indices = np.array(range(1, 11)), # Cash inflow timestamps
        inv_stage=np.array([175] * 4), # Budget for each investment stage
        stages=np.arange(1, 21).reshape((4, 5)) # Timelines for each stage
    ) -> None:

        self.projects = projects
        self.risk_weights = risk_weights
        self.risk_max = risk_max
        self.st_min = st_min
        self.wacc = wacc
        self.times = times
        self.cash_inflow_indices = cash_inflow_indices
        self.inv_stage = inv_stage
        self.stages = stages
```

TNPV

The **net present value** of project i which is launched at time unit t is given by Eq. (1)

$$\widetilde{NPV}_i = \left(\frac{-c_i}{(1+wacc)^{(\sum_{t\in T} t\times X_{it})}} + \sum_{n=1}^{10} \frac{\widetilde{in\_cash}_i(n)}{(1+wacc)^{(\sum_{t\in T} t\times X_{it})+n}} \right) \times \sum_{t\in T} X_{it}$$

where X_{it} is the **decision variable** such that $X_{it} = \begin{cases} 1 \text{ if project } i \text{ is launched at time } t \\ 0 \text{ otherwise} \end{cases}$

The **total net present value** $\widetilde{TNPV}$ of the project portfolio is the sum of $\widetilde{NPV}_i$ of all projects and is given by Eq. (2)

$$\widetilde{TNPV} = \sum_{i\in I} \widetilde{NPV}$$

The implementation is written **efficiently** in **matrix** form **without** using **for** loops.

```
# Calculate the total NPV given the decisions and projects cash inflows
def total_npv(self, decisions, projects_cash_inflows):
    projects_start_times = decisions @ self.times # get starting times (if any)
    projects_start_times_discount = (1 + self.wacc) ** projects_start_times # find discount amount because of late starting time
    projects_cash_inflows_discounts = projects_start_times_discount.reshape(-1, 1) @ \
                                  ((1 + self.wacc) ** self.cash_inflow_indices).reshape(1, -1) # add additional discount amount
                                                                                              # because of late cash inflows

    projects_discounted_cash_inflows = projects_cash_inflows / projects_cash_inflows_discounts # discount cash inflows
    costs = self.projects['c'].values / projects_start_times_discount # discount costs

    npv = costs + np.sum(projects_discounted_cash_inflows, axis=1) # calculate NPV for each project
    project_overall_decision = np.sum(decisions, axis=1) # find selected projects
    tnpv = npv @ project_overall_decision # calculate total NPV

    return tnpv

# Calculate the TNPV list for all simulated cash inflows
def all_tnpv(self, decisions, projects_cash_inflows):
    tnpvs = pd.Series(projects_cash_inflows).apply(lambda x: self.total_npv(decisions, x)) # calculate the TNPV list

    return tnpvs
```

VaR & CVaR

Two metrics widely used to quantify risk are **value-at-risk** (VaR) and **conditional value-at-risk** (CVaR). For **confidence** level $0 \le \alpha < 1$, $\mathrm{VaR}_\alpha(\widetilde{TNPV})$ is the $(1-\alpha)$-**quantile** of the $\widetilde{TNPV}$ distribution which is the **largest** value that ensures that the **probability** of obtaining a $TNPV$ **less** than this value is **lower** than $(1-\alpha)\epsilon(0,1)$.

$$\mathrm{VaR}_\alpha(\widetilde{TNPV}) = \max\{TNPV \in R : P(\widetilde{TNPV} < TNPV) \le (1-\alpha)\}$$

where, R is the support of probability distribution of $\widetilde{TNPV}$

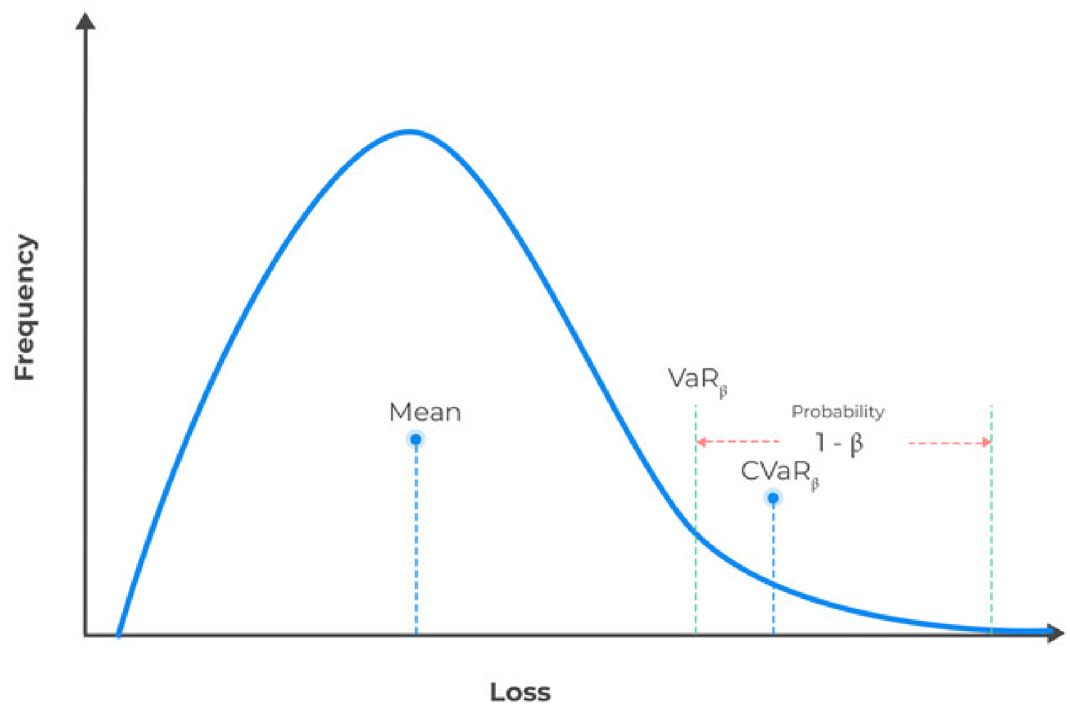
```
# Calculate the VaR for TNPV given the simulated cash inflows
def var_tnpv(self, decisions, projects_cash_inflows, alpha=0.95):
    tnpvs = self.all_tnpv(decisions, projects_cash_inflows) # get the TNPV list
    var_tnpv_ = tnpvs.sort_values().head(int((1 - alpha) * len(tnpvs))).iloc[-1] # calculate the VaR

    return var_tnpv_
```

CVaR at **confidence** level α (CVaR_α) is defined as the **expected** value of $\widetilde{TNPV}$ **smaller** than the $(1-\alpha)$-**quantile** of the probability distribution of $\widetilde{TNPV}$ as shown in Eq. (3).

$$\begin{aligned}\mathrm{CVaR}_\alpha(\widetilde{TNPV}) &= \mathbb{E}\left\{\widetilde{TNPV} \mid \widetilde{TNPV} \le \mathrm{VaR}_\alpha(\widetilde{TNPV})\right\} \\ &= \frac{1}{(1-\alpha)} \int_{-\infty}^{\mathrm{VaR}_\alpha(\widetilde{TNPV})} \mathrm{TNPV}\ \mathrm{F}(\widetilde{TNPV}) dTNPV\end{aligned}$$

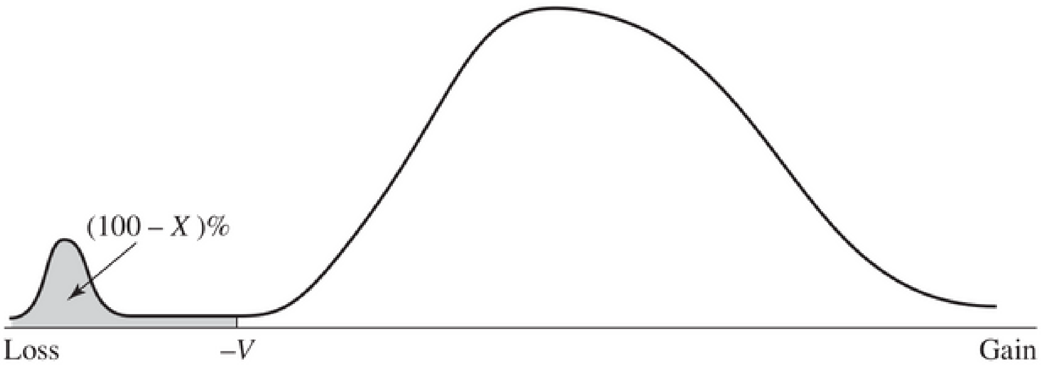
where $\mathrm{F}(\widetilde{TNPV})$ is the probability density function of the total net present value.

```
# Calculate the CVaR for TNPV given the simulated cash inflows
def cvar_tnpv(self, decisions, projects_cash_inflows, alpha=0.95):
    tnpvs = self.all_tnpv(decisions, projects_cash_inflows) # get the TNPV list
    cvar_tnpv_ = tnpvs.sort_values().head(int((1 - alpha) * len(tnpvs))).mean() # calculate the CVaR

    return cvar_tnpv_
```



The choice between VaR and CVaR depends on the specific needs and preferences of the investor or risk manager. Here are some considerations:

1. **Interpretability**: VaR provides a **straightforward** measure of the worst-case loss, which can be **easier** understand and communicate. On the other hand, CVaR provides **additional** information about the expected **magnitude** of losses beyond the VaR threshold.
2. **Risk tolerance**: If an investor wants to focus on the **extreme** tail risks and have a better understanding of the potential losses beyond the VaR level, CVaR may be more suitable. It captures the **tail risk** more **effectively** than VaR.
3. **Portfolio optimization**: CVaR is often **preferred** in portfolio optimization because it considers the **severity** of losses beyond the VaR level. It can help constructing portfolios that minimize the **expected losses** at the **tail** region.
4. **Computational complexity**: CVaR calculations are generally **more** computationally **intensive** than VaR calculations since they involve averaging the losses beyond the VaR threshold. VaR is **simpler** to calculate and interpret.
5. **Subadditivity**: This property checks that the risk measure for two portfolios after they have been merged should be no greater than the sum of their risk measures before they were merged. CVaR satisfies and VaR dissatisfies subadditivity.



A scenario where a large loss is more likely

In this project CVaR is chosen over VaR because we are interested the expected magnitude of losses beyond the VaR threshold.

Project Portfolio Selection and Scheduling Model

Models

Three project portfolio selection and scheduling models namely **risk-neutral (max_E)**, **risk-averse (max_CVaR)** and **combined compromise (max_E_CVaR)** are formulated. The objective functions of the three models are defined as follows.

- **max_E model**: Maximize the expected $TNPV$: maximize $E(\widetilde{TNPV})$

```
# Calculate the Expected TNPV given the simulated cash inflows
def expected_tnpv(self, decisions, projects_cash_inflows):
    tnpvs = self.all_tnpv(decisions, projects_cash_inflows) # get the TNPV list
    mean_tnpv = tnpvs.mean() # calculate the mean

    return mean_tnpv
```

- **max_CVaR model**: Minimize the risk of obtaining severe low $TNPV$: maximize $CVaR_\alpha(E(\widetilde{TNPV}))$
- **max_E_CVaR model**: Maximize the weighted sum of expected $TNPV$ and $CVaR_\alpha(\widetilde{TNPV})$: maximize $\left(\lambda E(\widetilde{TNPV}) + (1-\lambda) CVaR_\alpha(\widetilde{TNPV})\right)$ where, λ is the weightage given to E_TNPV.

```
# Calculate the weighted average of CVaR and Expectation for TNPV given the simulated cash inflows
def weighted_cvar_expected_tnpv(self, decisions, projects_cash_inflows, alpha=0.95, lambda_=0.5):
    tnpvs = self.all_tnpv(decisions, projects_cash_inflows) # get the TNPV list
    mean_tnpv = tnpvs.mean() # calculate the mean
    cvar_tnpv_ = tnpvs.sort_values().head(int((1 - alpha) * len(tnpvs))).mean() # calculate the CVaR

    weighted_cvar_expected_tnpv = lambda_ * mean_tnpv + (1 - lambda_) * cvar_tnpv_ # calculate the weighted average

    return weighted_cvar_expected_tnpv
```

Constraints

All optimizations are subject to **constraints** that are as follows:

- Equation (4) indicates that a project should be selected only once:

$$\sum_{t \in T} x_{it} \leq 1 \quad (\forall i \in I)$$

```
# Check the once selection constraint
def once_selection_constraint(self, decisions):
    project_overall_decision = np.sum(decisions, axis=1) # find selected projects
    constraint_satisfaction = not np.any(project_overall_decision > 1) # check if any project is selected more than once

    return constraint_satisfaction
```

- Equations (5), (6), (7) and (8) are stage budget constraints and indicate that the total investment in a stage should be less than or equal to the investment available in that stage.

$$\sum_{i \in I} c_i \sum_{t=1}^{5} x_{it} + o_{s_1} \leq \mathrm{Inv}_{s_1}$$

```
# Check the budget constraint for stage 1
def budget_constraint_stage_1(self, decisions, return_value=False):
    costs = np.abs(self.projects['c'].values) # get absolute costs
    remaining_budget = 0 # initialize unutilized investment overflowing to the next stage

    for i in range(1):
        stage = self.stages[i]
        stage_decisions = decisions[:, stage - 1] # get decisions at that stage
        stage_cost = np.sum(costs.reshape(1, -1) @ stage_decisions) # calculate the stage cost
```

```
        remaining_budget = remaining_budget + self.inv_stage[i] - stage_cost # find unutilized investment overflowing to the next stage

    if return_value:
        return remaining_budget

    constraint_satisfaction = remaining_budget >= 0 # check if the remaining budget is not negative

    return constraint_satisfaction
```

$$\sum_{i \in I} c_i \sum_{t=6}^{10} x_{it} + o_{s_2} \leq \text{Inv}_{s_2} + o_{s_1}$$

```
# Check the budget constraint for stage 2
def budget_constraint_stage_2(self, decisions, return_value=False):
    costs = np.abs(self.projects['c'].values) # get absolute costs
    remaining_budget = 0 # initialize unutilized investment overflowing to the next stage

    for i in range(2):
        stage = self.stages[i]
        stage_decisions = decisions[:, stage - 1] # get decisions at that stage
        stage_cost = np.sum(costs.reshape(1, -1) @ stage_decisions) # calculate the stage cost

        remaining_budget = remaining_budget + self.inv_stage[i] - stage_cost # find unutilized investment overflowing to the next stage

    if return_value:
        return remaining_budget

    constraint_satisfaction = remaining_budget >= 0 # check if the remaining budget is not negative

    return constraint_satisfaction
```

$$\sum_{i \in I} c_i \sum_{t=11}^{15} x_{it} + o_{s_3} \leq \text{Inv}_{s_3} + o_{s_2}$$

```
# Check the budget constraint for stage 3
def budget_constraint_stage_3(self, decisions, return_value=False):
    costs = np.abs(self.projects['c'].values) # get absolute costs
    remaining_budget = 0 # initialize unutilized investment overflowing to the next stage

    for i in range(3):
        stage = self.stages[i]
        stage_decisions = decisions[:, stage - 1] # get decisions at that stage
        stage_cost = np.sum(costs.reshape(1, -1) @ stage_decisions) # calculate the stage cost

        remaining_budget = remaining_budget + self.inv_stage[i] - stage_cost # find unutilized investment overflowing to the next stage

    if return_value:
        return remaining_budget

    constraint_satisfaction = remaining_budget >= 0 # check if the remaining budget is not negative

    return constraint_satisfaction
```

$$\sum_{i \in I} c_i \sum_{t=16}^{20} x_{it} \leq \text{Inv}_{s_4} + o_{s_3}$$

```
# Check the budget constraint for stage 4
def budget_constraint_stage_4(self, decisions, return_value=False):
    costs = np.abs(self.projects['c'].values) # get absolute costs
    remaining_budget = 0 # initialize unutilized investment overflowing to the next stage

    for i in range(4):
        stage = self.stages[i]
        stage_decisions = decisions[:, stage - 1] # get decisions at that stage
        stage_cost = np.sum(costs.reshape(1, -1) @ stage_decisions) # calculate the stage cost

        remaining_budget = remaining_budget + self.inv_stage[i] - stage_cost # find unutilized investment overflowing to the next stage

    if return_value:
        return remaining_budget

    constraint_satisfaction = remaining_budget >= 0 # check if the remaining budget is not negative

    return constraint_satisfaction
```

- Equation (9) indicates that the mandatory projects, P6 and P8, must be selected.

$$\sum_{t \in T} x_{6t} + \sum_{t \in T} x_{8t} = 2$$

```
# Check the mandatory project selection constraint
def mandatory_projects_constraint(self, decisions):
    project_overall_decision = np.sum(decisions, axis=1) # find selected projects
    constraint_satisfaction = np.sum(project_overall_decision[[5, 7]]) == 2 # check if the mandatory projects are selected

    return constraint_satisfaction
```

- Equation (10) indicates that the total strategic score of the portfolio should be greater than or equal to the minimum acceptable strategic alignment score.

$$\sum_{i \in I} st\_sc_i \sum_{t \in T} x_{it} \geq st\_\min$$

```
# Check the strategic scores constraint
def strategic_scores_constraint(self, decisions, return_value=False):
    strategic_scores = self.projects['st_sc'].values # find the strategic scores
    total_strategic_score = strategic_scores @ np.sum(decisions, axis=1) # calculate the overall strategic score

    if return_value:
        return total_strategic_score - self.st_min
```

```python
        constraint_satisfaction = total_strategic_score >= self.st_min # check if the overall strategic score is sufficient

        return constraint_satisfaction
```

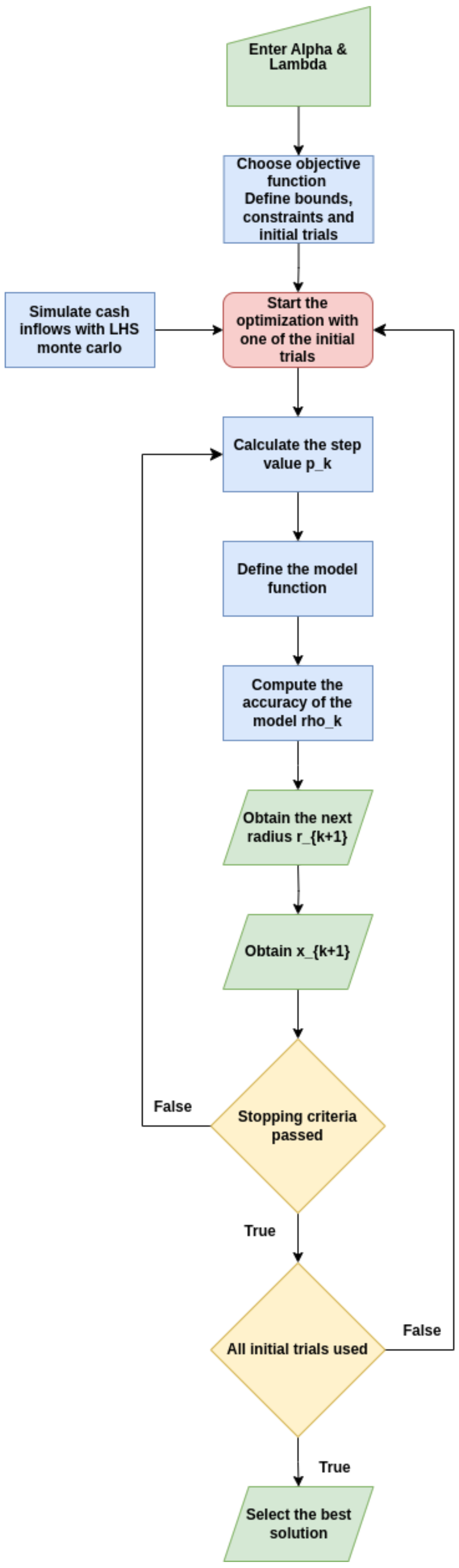
- Equation (11) indicates that the total risk of the portfolio should be less than or equal to the maximum acceptable risk.

$$
\begin{aligned}
& w_{\text{Tech\_risk}} \sum_{i \in I} Tech\_risk_i \sum_{t \in T} x_{it} + w_{\text{Sch\_risk}} \sum_{i \in I} Sch\_risk_i \sum_{t \in T} x_{it} \\
& + w_{EP\_risk} \sum_{i \in I} EP\_risk_i \sum_{t \in T} x_{it} + w_{\text{Org\_risk}} \sum_{i \in I} \text{Org\_risk}_i \sum_{t \in T} x_{it} \\
& + w_{\text{Stc\_risk}} \sum_{i \in I} Stc\_risk_i \sum_{t \in T} x_{it} \leq risk\_\text{max}
\end{aligned}
$$

```python
    # Check the maximum tolerated risk constraint
    def maximum_risk_constraint(self, decisions, return_value=False):
        project_risks = self.projects[['Tech_risk', 'Sch_risk', 'EP_risk', 'Org_risk', 'Stc_risk']].values # find the risk values
        total_risks = np.sum(decisions.T @ project_risks, axis=0) # calculate the overall risk for each risk
        final_risk = self.risk_weights @ total_risks # calculate the weighted overall risk

        if return_value:
            return self.risk_max - final_risk

        constraint_satisfaction = final_risk <= self.risk_max # check if the overall risk is tolerated

        return constraint_satisfaction
```

Solution Methodology

Simulation optimization is used as a solution methodology for the formulated project portfolio selection and scheduling model. It incorporates uncertainties in the input parameters and uses a combination of simulation and meta-heuristic search algorithm to optimize a static (e.g. mean, variance, VaR, CVaR) as an objective function.
The process starts with choosing α and λ to choose the **objective** function. Then, **200** project cash inflows are generated using **Latin Hypercube Sampling (LHS)**, from their **probability** distribution. The simulation generates a distribution of the **possible outcomes** of the objective function. Next, the **constraints** are defined and with an **initial** set of values of **decision variables**, **cash inflows** and the **constraint** functions are passed to the **optimizer**.
The original paper used an optimizing software that requires **paid subscribtion** so a **new** optimization procedure and methodology had to be used.
Using the python library **Scipy**, the `scipy.optimize.minimize` optimizer with **Trust Region Optimization Method** `method='trust-constr'` is utilized for the implementation of simulation optimization which **considers** the **constraints** during optimization process.
The process **stops** when the stopping **criteria** (e.g. **number of trial solutions**, **"gtol"** or **"xtol"** criteria of the optimizer function) is met. The trial **solution** that provides the **highest** value of the static of the objective function is selected as the solution of the problem.

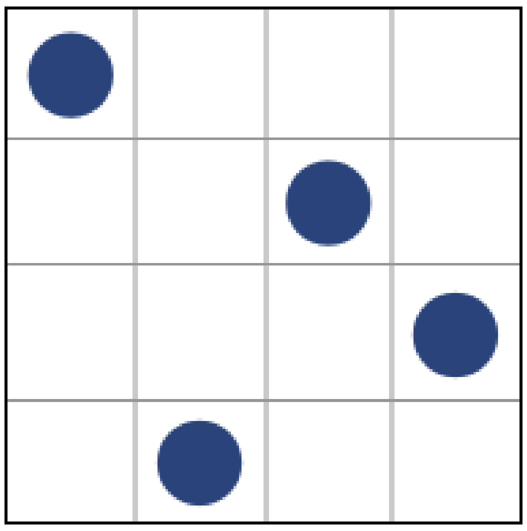


Steps of simulation optimization

Latin Hypercube Sampling (LHS)

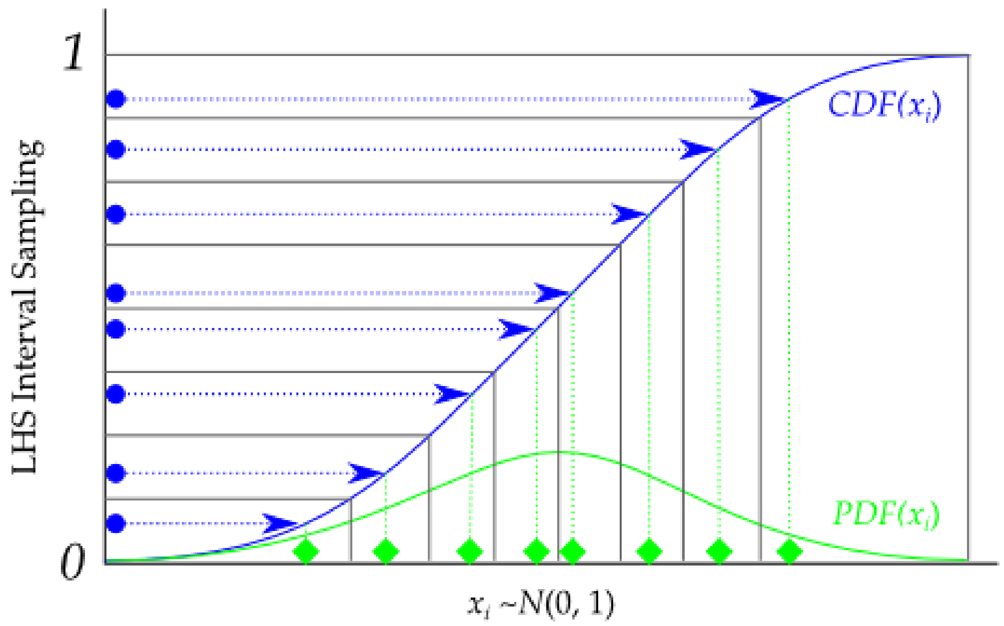
Latin hypercube sampling (LHS) is a statistical method for generating a **near-random** sample of parameter values from a multidimensional distribution.

The **idea** behind *LHS* is to **divide** the multidimensional parameter space into a number of intervals, and then to **randomly** select one value from each interval. This ensures that **each** interval is represented in the sample, and that the sample is spread out **evenly** over the parameter space.



Latin Hypercube Sampling

The samples obtained are sent to the **quantile** function of the **Gaussian** distribution with predefined parameters to complete the **monte carlo** simulation of the cash inflows.



Monte Carlo Simulation

```python
# Sample cash inflows at each state using Latin Hypercube Sampling and Normal distribution
def normal_latin_hypercube_monte_carlo(
    project_means,
    project_stds,
    n_samples=200,
    n_jobs=1,
    backend='loky'
):
    os.environ['JOBLIB_TEMP_FOLDER'] = '/tmp'

    def parallel_job(sample): # parallelize the sampling
        cash_inflow = np.zeros(project_means.shape)
        for i, j in itertools.product(range(project_means.shape[0]), range(project_means.shape[1])):
            project_std = project_stds[i, j]
            if project_std > 0:
                cash_inflow[i, j] = norm(loc=project_means[i, j], scale=project_stds[i, j]).ppf(sample) # use the provided distribution to sample cash inflow
            else:
                cash_inflow[i, j] = project_means[i, j] # if std. is zero, use the mean

        return cash_inflow

    cash_inflows = []
    lhd = qmc.LatinHypercube(d=1, seed=1).random(n=n_samples) # sample numbers in [0, 1] to obtain final samples from the ppf distribution

    cash_inflows = Parallel(n_jobs=n_jobs, backend=backend)(delayed(parallel_job)(sample) # find cash inflows
                                                            for sample in lhd)

    return cash_inflows
```

Advantages

- It ensures that **each** interval in the parameter space is represented in the sample. This is important because it helps to ensure that the sample is **not biased** towards any particular region of the parameter space.
- *LHS* is relatively **easy** to implement.
- *LHS* is a **deterministic** method, which means that the **same** sample will be generated each time the code is run.

Disadvantages

- it can be **inefficient** for **high-dimensional** problems.
- *LHS* can be **sensitive** to the choice of the **number of intervals**. If the number of intervals is too **small**, then the sample may **not** be spread out **evenly** over the parameter space. If the number of intervals is too **large**, then the sample may be too large and **computationally expensive** to generate.

Compared to other sampling methods:

Method	Description	Pros	Cons
LHS	A statistical method for generating a near-random sample of parameter values from a multidimensional distribution.	- Ensures that each dimension of the sample space is evenly sampled. - Can be more efficient than MCS for small sample sizes.	- Can be more difficult to implement than MCS. - Can be less efficient than MCS for large sample sizes.
MCS	A method for generating a random sample of parameter values from a multidimensional distribution.	- Simple to implement. - Can be used with any probability distribution.	- Can be less efficient than LHS for small sample sizes. - Can be less accurate than LHS for some distributions.
Grid sampling	A method for generating a sample of parameter values from a multidimensional distribution by evaluating the function at a regular grid of points.	- Simple to implement. - Can be used with any probability distribution.	- Can be less efficient than MCS or LHS. - Can be less accurate than MCS or LHS for some distributions.

Trust Region Methods

Trust-region methods are a class of numerical optimization methods that are used to solve **nonlinear** optimization problems. The basic idea of trust-region methods is to approximate the objective function within a **trust region** around the current iterate. The trust region is a small region of the search space where the objective function is assumed to be **well-behaved**. The trust-region methods then use this approximation to find the **next** iterate that **minimizes** the objective function within the trust region subject to **constraints**.

Overview of the Trust-Region Approach

Suppose we wish to **minimize** a function f. Given some particular point x_k in the domain of f, how do we select a new point x_{k+1} that better minimizes the function? A **line-search algorithm** solves this sub-problem by first choosing a search **direction** d_k (often related to the gradient of f), and then a **step** length α_k so as to minimize f along the direction d_k. The next point, then, is simply

$$x_{k+1} := x_k + \alpha_k d_k.$$

A **trust-region algorithm** approximates the function f with some **simpler** function m_k (called the **model function**) in a **neighborhood** of x_k. The model m_k will likely **not** be a good approximation for f over the **entire** domain, and so we must **restrict** our attention to a **ball** of radius r_k centered at the point x_k, inside of which m_k is **reasonably** close to f. We then **minimize** m_k over this ball subject to **constraints**, and set x_{k+1} equal to this minimizer. That is, we compute x_{k+1} by solving the sub-problem

$$x_{k+1} := \underset{x \in B(x_k, r_k)}{\operatorname{argmin}} \, m_k(x).$$

The ball $B\left(x_k, r_k\right)$ is called the **trust region** because we trust that the model function m_k gives a reasonably accurate approximation of f on this region.

We define x_k as an **array** of length 20 where for each element located at location i, its value $x_{ki} \in [0, 20]$ specifies the **starting time** of the Pi project ($x_{ki} = 0$ means the project is **not** selected). For example:

```
x0 = [0,  3,  0,  0,  0,  20,  1,  20,  0,  0,  0,  18,  11,  0,  6,  0,  12,  0,  0,  0]
```

To evaluate each trial, x_k we should convert it to the **decision** variables matrix X_k as the input for our objective function.

```
# Convert project start time array to the decision matrix
def convert_result_to_decisions(project_decisions):
    decisions = [] # initialize projects decisions

    for random_number in project_decisions:
        decision = np.zeros(20) # initialize project decision

        if int(random_number) > 0:
            decision[int(random_number) - 1] = 1 # if a project is selected, its start time index has value 1

        decisions.append(decision)

    decisions = np.vstack(decisions)

    return decisions
```

An example of the result is:

```
X0 = np.array(
    [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.],
```

```
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0.]]
)
```

The Model Function

The model function is commonly taken to be a **linear** or **quadratic** approximation of f based on its **Taylor Series** expansion about the point x_k. In the **linear** case, our model function has the form

$$m_k(x) = f(x_k) + (x - x_k)^T \nabla f(x_k).$$

In the **quadratic** case, we simply add on a quadratic term to obtain

$$m_k(x) = f(x_k) + (x - x_k)^T \nabla f(x_k) + \frac{1}{2}(x - x_k)^T H_k (x - x_k),$$

where H_k is the **Hessian** matrix of f at x_k, or some approximation thereof. Given a trust region with **radius** r_k, note that our sub-problem can be written in the following way:

$$\begin{aligned} x_{k+1} &= \underset{x \in B(x_k, r_k)}{\operatorname{argmin}} \, m_k(x) \\ &= x_k + p_k, \end{aligned}$$

where

$$p_k = \underset{\|p\| < r_k}{\operatorname{argmin}} \left\{ f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T H_k p \right\}.$$

p_k is called a **step**. Also we define p to be

$$p = x - x_k$$

We define the **model** function:

$$m_k(p) = f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T H_k p,$$

The Trust-Region Radius

A **crucial** aspect of trust-region algorithms is the choice of **radius** r_k. If r_k is too **small**, then the algorithm will make **slow** progress toward the minimizer of f. If r_k is too **large**, the model function will be a **poor fit** for the objective function f, and the next iterate x_{k+1} may **fail** to decrease f. A reasonably **robust** trust-region algorithm must therefore be able to **adaptively** choose the trust-region radius.

Our strategy for choosing an appropriate radius r_{k+1} for the $(k+1)$-th iterate involves evaluating the **accuracy** of the **model** function at the k-th iterate. If the model was **accurate** and a **large** step was taken, we can optimistically choose r_{k+1} to be **larger** than r_k in the hopes of achieving **faster** convergence. To prevent the radius from growing too large, we set an overall **bound** $r_{\max}$ on the trust-region radii. If the model was very **inaccurate**, we make r_{k+1} **smaller** than r_k, since the model function can't be trusted over such a large region. If the model was **neither** particularly accurate nor inaccurate, we simply choose $r_{k+1} = r_k$.

We measure the **accuracy** of the model by the **ratio** of the **actual** reduction to the **predicted** reduction in the objective function.

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}.$$

The closer ρ_k is to 1 , the more **accurate** the model. Note that if ρ_k is **negative** or below a certain positive **threshold** η, then the point $x_k + p_k$ is a **poor** improvement over x_k (and perhaps is worse). In this case, we **reject** the new point and set $x_{k+1} = x_k$.

The Trust-Region Algorithm

We now **combine** the two steps of minimizing the model function and choosing the trust-region radius to build the algorithm. In practice, we **halt** the algorithm once $\|\nabla f(x_k)\|$ is less than some **threshold** value.

Algorithm 1.1 Trust-Region Algorithm

procedure TRUST-REGION ALGORITHM
  Choose initial point x_0, initial radius r_0, and threshold $\eta \in [0, 0.25)$.
  while $\|\nabla f(x_k)\| > tol$ **do**
    Calculate p_k by solving the sub-problem in Equation 1.1.
    Compute ρ_k.
    if $\rho_k < 0.25$ **then**
      $r_{k+1} = 0.25 r_k$
    else
      if $\rho_k > 0.75$ and $\|p_k\| = r_k$ **then**
        $r_{k+1} = \min(2r_k, r_{max})$
      else
        $r_{k+1} = r_k$
    if $\rho_k > \eta$ **then**
      $x_{k+1} = x_k + p_k$
    else
      $x_{k+1} = x_k$

Trust-Region Algorithm

In this project we run the optimization for **multiple initial trials** and select the **best** solution of them all. On top of that, we pass our problem **constraints** to the function to be considered on top of the trust region constraints.

```
# run the optimization with the defined constraints and all possible initial points
for x00 in tqdm(initial_points.values()):
    result = minimize(objective, x00, method='trust-constr',
                constraints=[NonlinearConstraint(non_linear_budget_constraint_stage_1, 0, np.inf, keep_feasible=True),
                             NonlinearConstraint(non_linear_budget_constraint_stage_2, 0, np.inf, keep_feasible=True),
                             NonlinearConstraint(non_linear_budget_constraint_stage_3, 0, np.inf, keep_feasible=True),
                             NonlinearConstraint(non_linear_budget_constraint_stage_4, 0, np.inf, keep_feasible=True),
                             NonlinearConstraint(non_linear_strategic_scores_constraint, 0, np.inf, keep_feasible=True),
                             NonlinearConstraint(non_linear_maximum_risk_constraint, 0, np.inf, keep_feasible=True)
```

```
        ],
        options={'verbose' : 1, 'maxiter' : 1000}, bounds=bounds, callback=callback)
```

Stopping Criteria

- `maxiter` : Maximum number of algorithm iterations. Default is 1000. We selected **200**,
- `gtol` : Tolerance for termination by the norm of the Lagrangian gradient. The algorithm will terminate when both the infinity norm (i.e., max abs value) of the Lagrangian gradient and the constraint violation are smaller than `gtol` . Default is **1e-8**.
- `xtol` : Tolerance for termination by the change of the independent variable. The algorithm will terminate when `tr_radius < xtol` , where `tr_radius` is the radius of the trust region used in the algorithm. Default is **1e-8**.

Comparison

The trust-region methods are a powerful class of optimization methods that can be used to solve a wide variety of nonlinear optimization problems. However, they can be computationally expensive, especially for large-scale problems. Compared to other optimization methods:

Method	Description	Pros	Cons
Trust-region methods	A class of numerical optimization methods that are used to solve nonlinear optimization problems. They are based on the idea of approximating the objective function within a trust region around the current iterate.	- Robust to ill-conditioned problems. - Can be used with non-convex objective functions.	- Can be computationally expensive, especially for large-scale problems.
Gradient descent methods	A class of numerical optimization methods that are based on the gradient of the objective function. They iteratively update the iterate in the direction of the negative gradient.	- Simple to implement. - Efficient for convex problems.	- Can be slow to converge for non-convex problems. - Can be trapped in local minima.
Newton methods	A class of numerical optimization methods that are based on the Hessian of the objective function. They iteratively update the iterate in the direction of the negative gradient, scaled by the inverse Hessian.	- Can be very efficient for convex problems. - Can escape from local minima.	- Can be difficult to implement for large-scale problems. - Sensitive to the condition number of the Hessian.
Conjugate gradient methods	A class of numerical optimization methods that are based on the gradient of the objective function. They iteratively update the iterate in the direction of the negative gradient, using a conjugate direction search.	- Efficient for convex problems. - Robust to ill-conditioned problems.	- Can be slow to converge for non-convex problems.

Results

The aforementioned simulation methodology is applied in **risk-neutral (max_E)**, **risk-averse (max_CVaR)** and **combined compromise (max_E_CVaR)** models.

- The **max_E** mode is run only **once** which yields maximum objective function $E(\widetilde{TNPV})$ of the selected and scheduled project portfolio and calculates $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$and $\mathrm{VaR}_\alpha(\widetilde{TNPV})$ corresponding to multiple confidence levels $\alpha \in \{90\%, 92.5\%, 95\%, 97.5\%, 99\%\}$ from the obtained probability distribution of $\widetilde{TNPV}$.
- The **max_CVaR** model was run **5** times for confidence levels $\alpha \in \{90\%, 92.5\%, 95\%, 97.5\%, 99\%\}$. Each model yields maximum objective function $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ for the corresponding a value and calculates $E(\widetilde{TNPV})$ of the selected project portfolio.
- The **max_E_CVaR** model is run **5×5** times for all combinations of confidence levels $\alpha \in \{90\%, 92.5\%, 95\%, 97.5\%, 99\%\}$ and lambda values $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Each model yields the maximum objective function $\left(\lambda E(\widetilde{TNPV}) + (1-\lambda) CVaR_\alpha(\widetilde{TNPV})\right)$ for corresponding α and λ values.

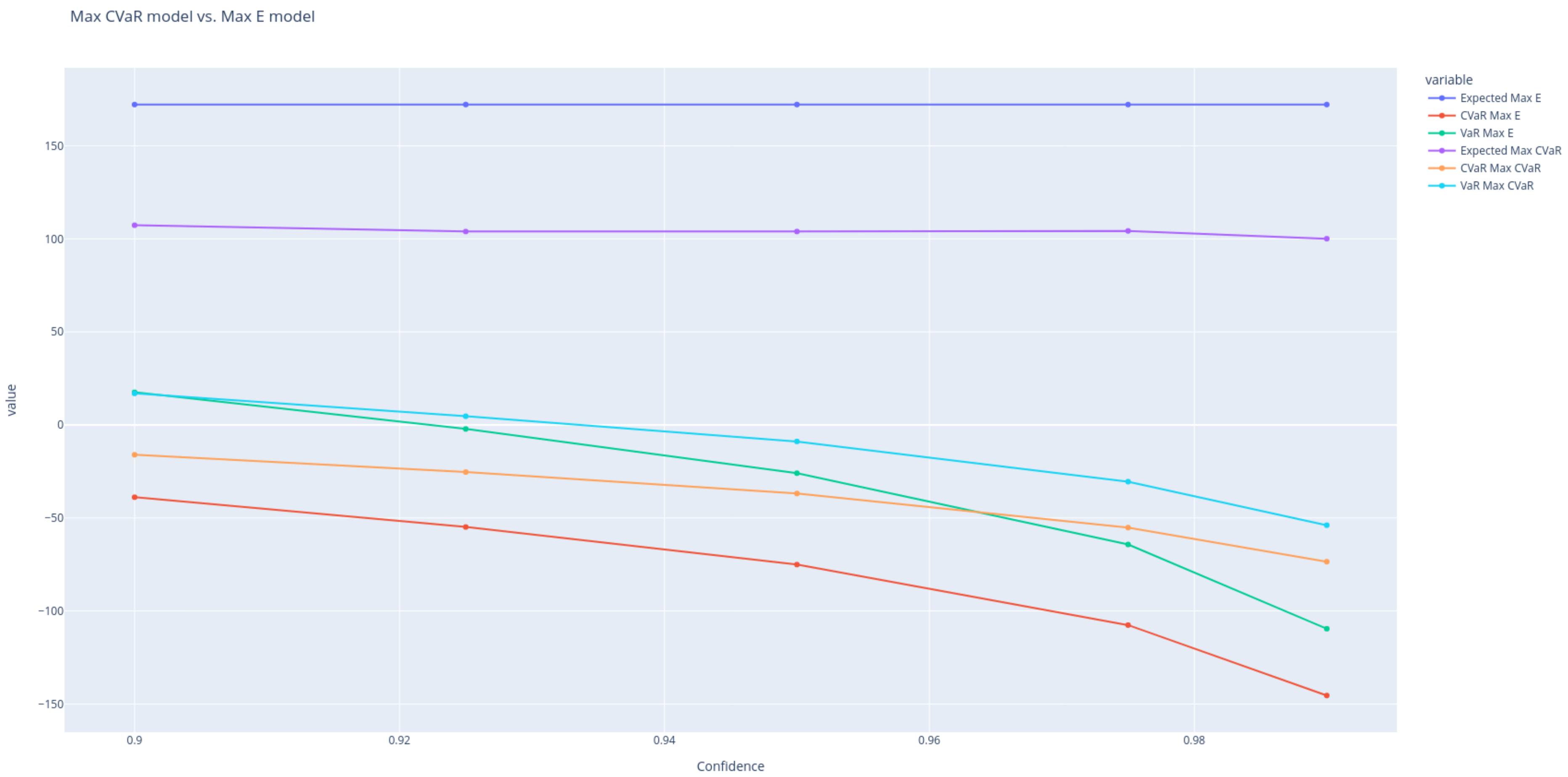
Table 4 presents the selected projects and their launch dates (time units t).

	Alpha	Lambda	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	Derivative	Platform	Breakthrough	R&D	Total
0	1	1	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
1	0.9	0	0	11	0	1	0	19	2	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
2	0.9	0.1	0	11	0	1	0	19	3	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
3	0.9	0.3	0	2	0	0	0	19	1	19	0	0	0	17	11	0	6	0	12	0	0	0	4	2	2	0	8
4	0.9	0.5	0	3	0	0	0	20	1	19	0	0	0	17	11	0	6	0	11	0	0	0	4	2	2	0	8
5	0.9	0.7	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
6	0.9	0.9	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
7	0.925	0	0	11	0	1	0	19	2	19	0	0	0	0	6	0	16	0	11	0	0	0	5	1	2	0	8
8	0.925	0.1	0	11	0	1	0	19	2	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
9	0.925	0.3	0	10	0	1	0	19	2	19	0	0	0	0	6	0	16	0	10	0	0	0	5	1	2	0	8
10	0.925	0.5	0	3	0	0	0	20	1	19	0	0	0	17	11	0	6	0	11	0	0	0	4	2	2	0	8
11	0.925	0.7	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
12	0.925	0.9	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
13	0.95	0	0	11	0	1	0	19	2	19	0	0	0	0	6	0	16	0	11	0	0	0	5	1	2	0	8
14	0.95	0.1	0	11	0	1	0	19	2	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
15	0.95	0.3	0	10	0	1	0	19	3	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
16	0.95	0.5	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
17	0.95	0.7	0	2	0	0	0	19	1	19	0	0	0	17	11	0	6	0	11	0	0	0	4	2	2	0	8
18	0.95	0.9	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
19	0.975	0	0	10	0	1	0	19	2	19	0	0	0	0	6	0	16	0	11	0	0	0	5	1	2	0	8
20	0.975	0.1	0	11	0	1	0	19	2	19	0	0	0	0	6	0	16	0	11	0	0	0	5	1	2	0	8
21	0.975	0.3	0	10	0	1	0	19	3	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
22	0.975	0.5	0	2	0	0	0	20	1	20	0	0	0	18	11	0	6	0	11	0	0	0	4	2	2	0	8

	Alpha	Lambda	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	Derivative	Platform	Breakthrough	R&D	Total
23	0.975	0.7	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
24	0.975	0.9	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
25	0.99	0	0	11	0	1	0	19	2	19	0	0	0	0	6	0	17	0	11	0	0	0	5	1	2	0	8
26	0.99	0.1	0	10	0	1	0	19	2	19	0	0	0	0	6	0	17	0	11	0	0	0	5	1	2	0	8
27	0.99	0.3	0	11	0	1	0	19	2	19	0	0	0	0	5	0	16	0	11	0	0	0	5	1	2	0	8
28	0.99	0.5	0	3	0	0	0	20	1	20	0	0	0	18	11	0	6	0	12	0	0	0	4	2	2	0	8
29	0.99	0.7	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8
30	0.99	0.9	0	1	0	0	0	20	1	20	0	0	0	1	16	0	11	0	16	0	0	0	4	2	2	0	8

Table 6 presents the values of $\mathrm{VaR}_\alpha(\widetilde{TNPV})$, $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ and $E(\widetilde{TNPV})$ of the selected and scheduled project portfolio corresponding to the aforementioned confidence levels for **max_E** and **max_CVaR** models.

Confidence	Expected Max E	CVaR Max E	VaR Max E	Expected Max CVaR	CVaR Max CVaR	VaR Max CVaR
0.9	172.204	-38.8613	17.5415	107.325	-16.0284	16.9353
0.925	172.204	-54.8631	-2.16216	103.952	-25.3467	4.66283
0.95	172.204	-75.0667	-25.9465	103.952	-36.8512	-8.88066
0.975	172.204	-107.592	-64.2623	104.276	-55.1984	-30.5016
0.99	172.204	-145.442	-109.557	100.093	-73.51	-53.8978



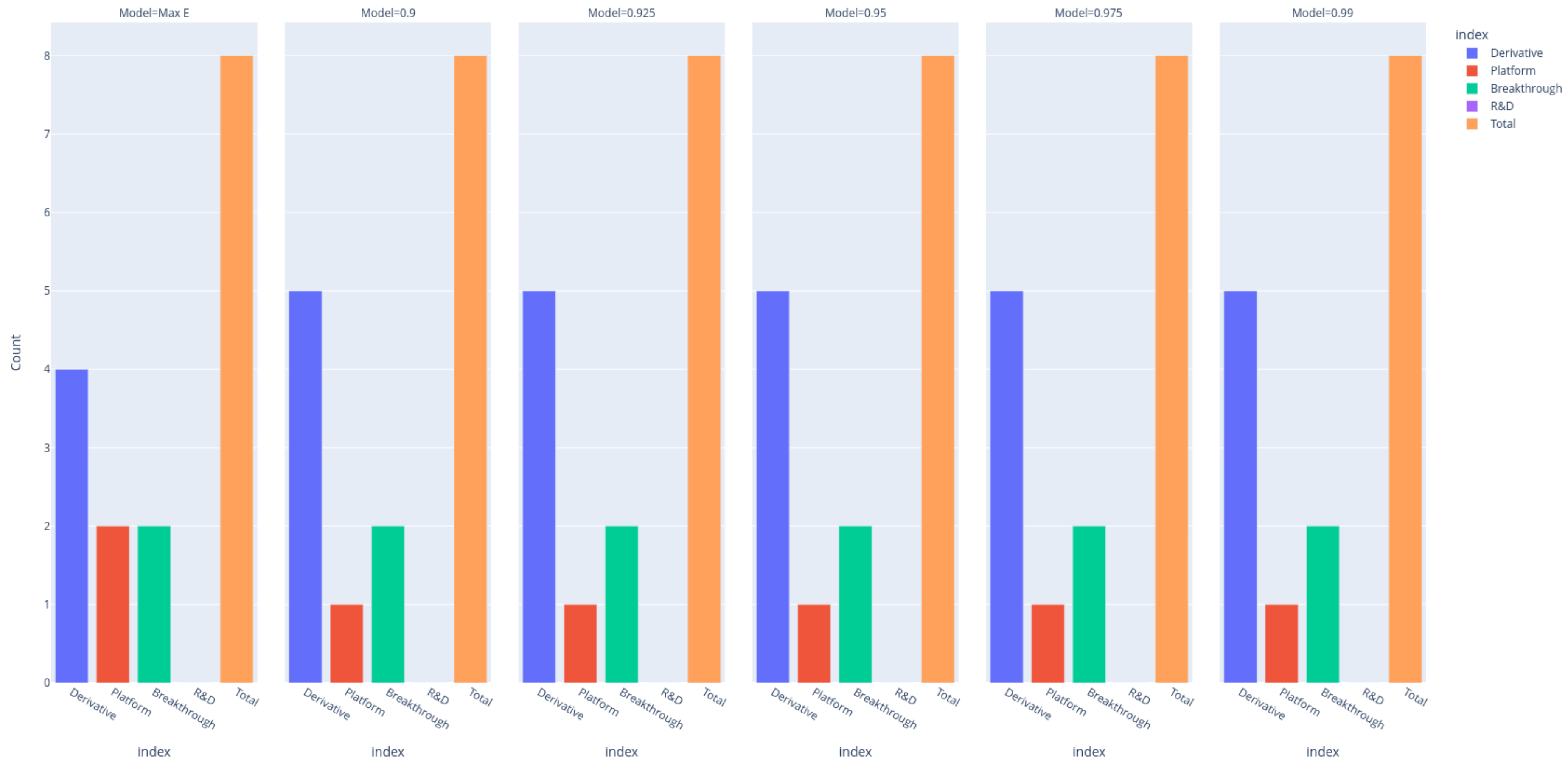
Max CVaR model vs. Max E model

Figure above reveals that the **max_E** model yields $E(\widetilde{TNPV})$ greater than to that of the **max_CVaR** model. Moreover, the risk measures $\mathrm{VaR}_\alpha(\widetilde{TNPV})$and $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ are higher for the **max_CVaR** model than that of **max_E** model. This indicates that the **max_CVaR** has **less risk** in expense of **less returns**. This ensures that the lowest $TNPV$ in the **worst scenario** is maximized to the greatest extent possible even when the confidence levels are low.

Further, Figure above shows that the differences in the values of $\mathrm{VaR}_\alpha(\widetilde{TNPV})$, $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ and $E(\widetilde{TNPV})$ in **max_CVaR** model and **max_E** model are **lower** for **higher** confidence levels $\alpha \in \{95\%, 97.5\%, 99\%\}$ than for lower confidence levels $\alpha \in \{90\%, 92.5\%\}$. This finding evidences the advantage of using the **max_CVaR** model in risk averse decision making at **lower** confidence levels.

Figure bellow shows that the **max_CVaR** model selected total number of projects that is equal to that of the **max_E** model. In all **max_CVaR** models **(derivative=5, platform =1, breakthrough=2)** the composition of the selected project portfolio with respect to number of projects in each category is different to that of the **max_E** model **(derivative=4 ,platform=2, breakthrough=2)** in the number of derivative and platform projects. The above observations demonstrate that, the **max_CVaR** reduces risk by increasing the number of derivative projects and reducing the number of platform projects.

Number of projects selected in each category



Number of projects selected in each category

As shown in **Table 4**,

- The composition that is selected by **both** models contains:

1	2	3	4	5	6	7
P2	P6	P7	P8	P13	P15	P17

- The composition that is selected by **both** models contains **none** of the following

1	2	3	4	5	6	7	8	9	10	11
P1	P3	P5	P9	P10	P11	P14	P16	P18	P19	P20

- The projects are **scheduled** to launch on **average** at times:

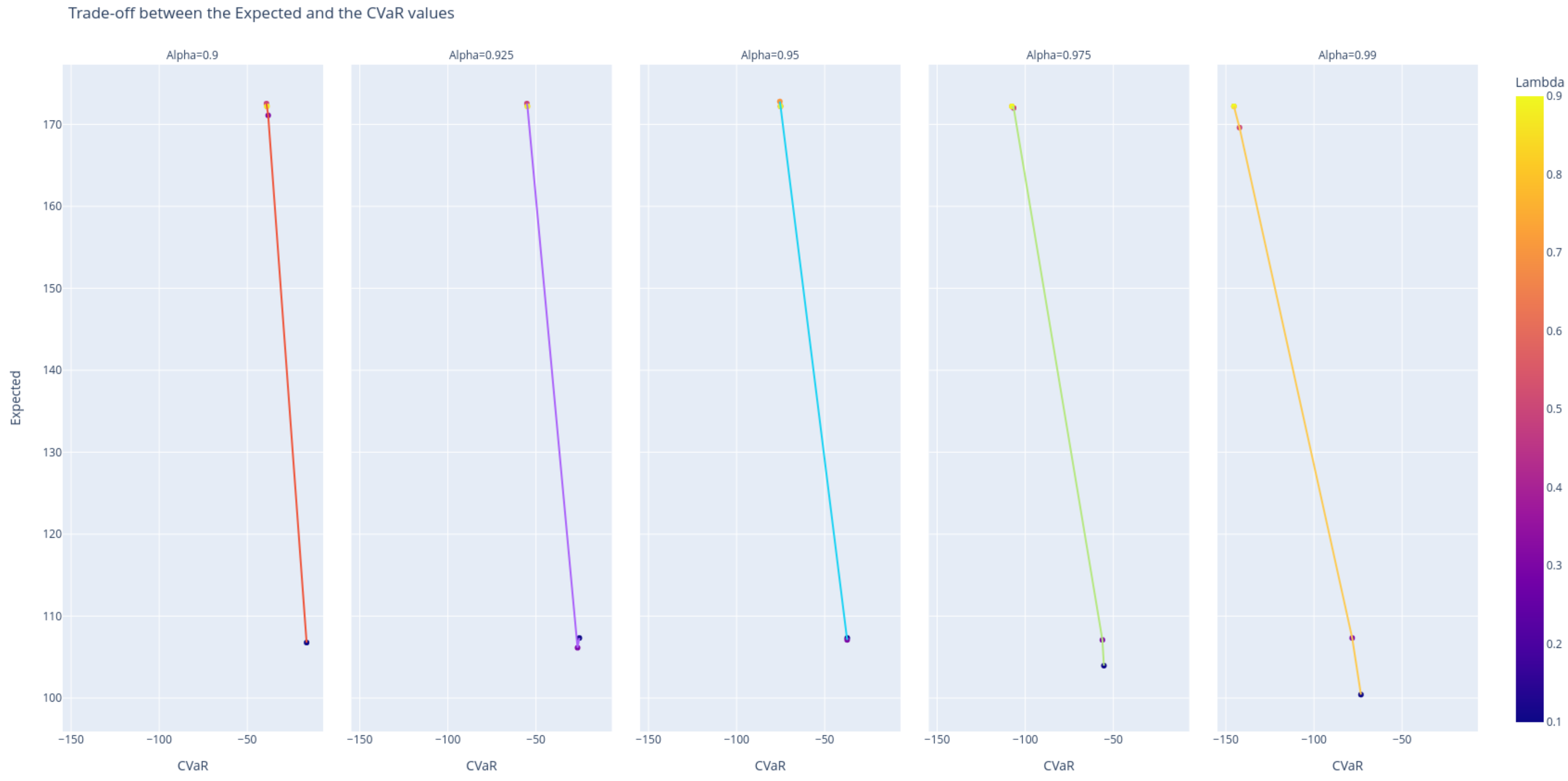
P2	P4	P6	P7	P8	P12	P13	P15	P17
9.16667	0.833333	19.1667	1.83333	19.1667	0.166667	7.5	15.3333	11.8333

- $P2$ **"setting up procurement and supplier management system"** a **derivative** project, is selected by all the models and is scheduled to be launched on average at time **5.6**.
- $P6$ **"Setting up effluent treatment plant"** and $P8$ **"Corporate social responsibility project"** are **mandatory** projects and are scheduled to be started mostly on the **last** stage i.e. on average at time **19.2**, as they generate **no cash inflows** but require initial **investment**.
- $P13$ **"Domestic market expansion"** a **platform** project is selected by all the models and is scheduled to be launched by most of the models in the **intermediate** stages i.e. on average at time **7.5**.
- $P17$ **"Introduction of sports drink"** a **breakthrough** project is selected by all the models and is scheduled to be launched by most of the models in the **later** stages i.e. on average at time **11.8**.
- The aforementioned project selection demonstrates that the obtained portfolios contain **at least one** project from **each** project category **(derivative, platform and breakthrough)**.

The table and figure bellow displays the **trade-off** between $E(\widetilde{TNPV})$ and $\mathrm{CVaR}_\alpha(\widetilde{TNPV})$ as weightage factor λ is varied for each value of α.

	Alpha	Lambda	Expected	CVaR
0	0.9	0.1	106.767	-16.3332
1	0.9	0.3	171.096	-38.1531
2	0.9	0.5	172.54	-39.1209
3	0.9	0.7	172.204	-38.8613
4	0.9	0.9	172.204	-38.8613
5	0.925	0.1	107.325	-25.3804
6	0.925	0.3	106.139	-26.4237
7	0.925	0.5	172.54	-55.168
8	0.925	0.7	172.204	-54.8631
9	0.925	0.9	172.204	-54.8631
10	0.95	0.1	107.325	-37.1881
11	0.95	0.3	107.091	-37.258
12	0.95	0.5	172.204	-75.0667
13	0.95	0.7	172.789	-75.4642
14	0.95	0.9	172.204	-75.0667
15	0.975	0.1	103.952	-55.3723
16	0.975	0.3	107.091	-56.2455
17	0.975	0.5	171.998	-106.574

	Alpha	Lambda	Expected	CVaR
18	0.975	0.7	172.204	-107.592
19	0.975	0.9	172.204	-107.592
20	0.99	0.1	100.417	-73.3565
21	0.99	0.3	107.325	-78.3177
22	0.99	0.5	169.61	-142.285
23	0.99	0.7	172.204	-145.442
24	0.99	0.9	172.204	-145.442



Trade-off between the Expected and the CVaR values

A decision maker who is **neutral** towards risk will choose the solution with higher expected total net present value $E(\widetilde{TNPV})$. By contrast, a risk **averse** decision maker will select the solution with a higher $\mathrm{CVaR}_{\alpha}(\widetilde{TNPV})$. The trade-off table shows that:

Alpha	Risk Neutral Lambda	Risk Neutral Expected	Risk Neutral CVaR	Risk Averse Lambda	Risk Averse Expected	Risk Averse CVaR
0.9	0.5	172.54	-39.1209	0.1	106.767	-16.3332
0.925	0.5	172.54	-55.168	0.1	107.325	-25.3804
0.95	0.7	172.789	-75.4642	0.1	107.325	-37.1881
0.975	0.7	172.204	-107.592	0.1	103.952	-55.3723
0.99	0.7	172.204	-145.442	0.1	100.417	-73.3565

The results obtained from the **max_E_CVaR** model enable decision makers to select and schedule project portfolio according to their risk appetite and the weightage they attribute to the risk measure $\mathrm{CVaR}_{\alpha}(\widetilde{TNPV})$ and the expected measure $E(\widetilde{TNPV})$.

Project portfolios are considered as strategic weapons for an organization's success. Thus project portfolio selection and scheduling optimization is critical for organizations. The present study evaluates strategic alignment scores and risk scores of the identified projects of a dairy firm and ensures that the selected project portfolio aligns with the strategic goals of the organization and is least exposed to risks.