Quantum Open Development Ecosystem (QODE) Optimization Division Specification Document

Q-STAR (Quantum STrategic industry Alliance for Revolution)

May 29, 2025

Contents

1 Overview **2**

2 System architecture and optimization steps **4**
(1)Create decision variables . . . 4
(2) Create objective function . . . 4
(3) Construct constraints . . . 4
(4) Formulate optimization model . . . 4
(5) Specify solver client attributes . . . 4
(6) Execute solver . . . 4

3 Decision Variables **6**
3.1 Variable Generator . . . 6
3.2 Methods and Arguments . . . 6
3.2.1 scalar() . . . 6
3.2.2 array() . . . 6
3.3 Get variable information . . . 7
3.4 Example code . . . 7

4 Polynomials **8**
4.1 Arithmetic of polynomial . . . 9
4.2 Arithmetic of polynomial arrays . . . 9
4.3 Attributes and methods of polynomial . . . 9
4.4 Example code . . . 11

5 Constraints **12**
5.1 Fix values . . . 12
5.2 Limit the range of variable values . . . 12
5.3 Limit the range of polynomial values . . . 12
5.3.1 Equality constraints . . . 12
5.3.2 Inequality constraints . . . 12
5.3.3 Constraint list . . . 12

5.3.4 Constraint weights . . . 13
5.4 Example code . . . 15

6 Model **16**
6.1 Model class . . . 16
6.2 Model attributes . . . 16
6.3 Example code . . . 18

7 Solver Client **19**

8 Solver Execution **19**
8.1 Solve function . . . 19
8.2 Retrieving the result . . . 19
8.3 Example code . . . 20

9 References **21**
9.1 Algebraic classes and numerics . . . 21
9.1.1 SDK.**Poly** class . . . 21
9.1.2 SDK.**PolyArray** class . . . 21
9.1.3 SDK.**Variable** class . . . 21
9.1.4 SDK.**VariableGenerator** class . . . 22
9.1.5 SDK.**sum()** function . . . 22
9.1.6 SDK.**VariableType** enum class . . . 22
9.2 Constraints . . . 22
9.2.1 SDK.**Constraint** class . . . 22
9.2.2 SDK.**ConstraintList** class . . . 23
9.2.3 Functions . . . 23
9.3 Model classes and functions . . . 23
9.3.1 SDK.**Model** class . . . 23
9.4 Solve (Solve classes and functions) . . . 24
9.4.1 SDK.**Result.Solution** class . . . 24
9.4.2 SDK.**Result** class . . . 24
9.4.3 SDK.Result.Values class . . . 24

1 Overview

This document describes an interface specification of a software development kit, hereinafter referred to as "the SDK," which is designed to solve a wide variety of combinatorial optimization problems.
This SDK is part of a software ecosystem designed to uniformly handle quantum computers including quantum annealing machines and quantum gate computers, shown in Fig.1. While its primary application currently targets mathematical optimization problems, it is expected to encompass quantum chemistry calculations and other applications in the future.

The specification will cover the following items.

- Assumed system architecture

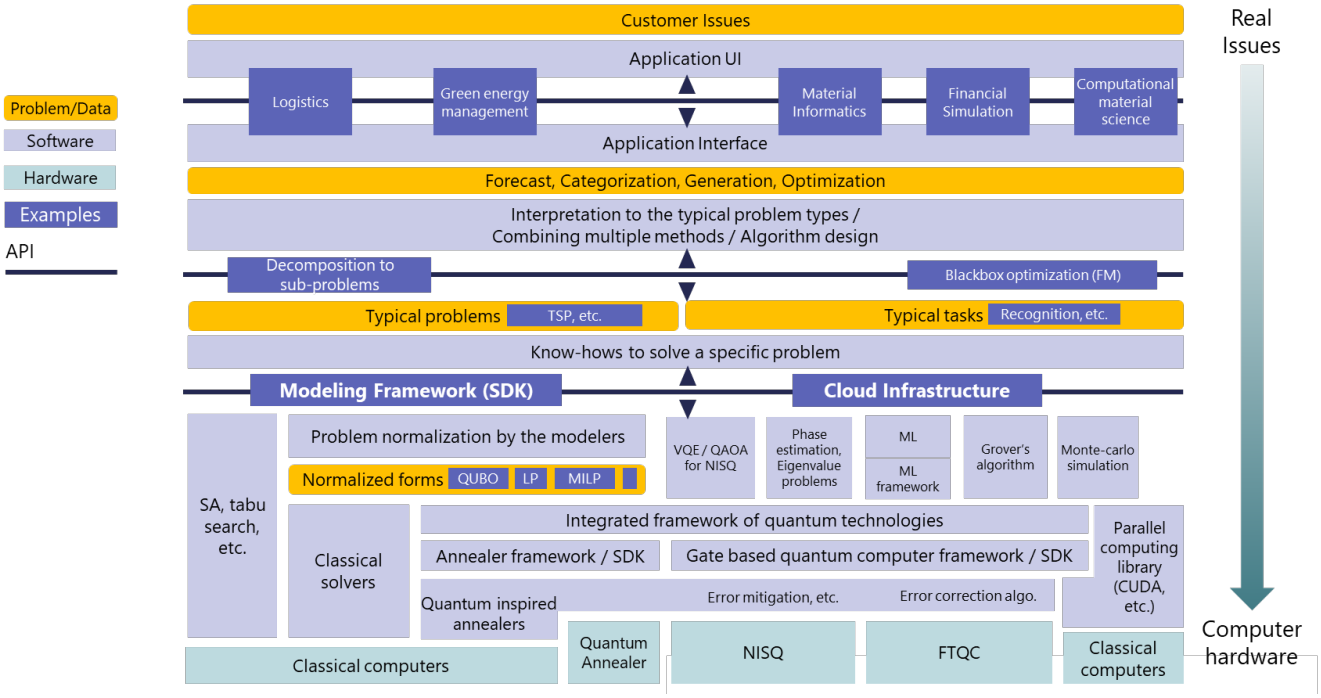

Figure 1: A schematic ecosystem for quantum computing

- Process of creating (formulating) a model and obtaining a solution
- Specification of arguments and return values of functions and classes required for formulation
- Specifications of processing performed by functions and class methods

On the other hand, the specification will NOT cover the following items.

- Algorithms and hardware specifications for calculations
- Non-Python library specifications

The SDK assumes Python as its runtime environment. Therefore, although attributes are described in the documentation, it is acceptable to implement them as properties.

2 System architecture and optimization steps

This specification assumes the following system architecture, shown in Fig.2.

The SDK MUST be able to perform optimization by formulating a combinatorial optimization problem using the following steps:

(1)Create decision variables

First, create a generator (**VariableGenerator**) that creates decision variables. Next, create a variable (**Poly**) or an array of variables (**PolyArray**) using the **VariableGenerator**.

(2) Create objective function

Define the objective function (**Poly**) using the variables generated by the **VariableGenerator**.

(3) Construct constraints

Construct constraints (**Constraint**) from polynomials using constraint creation functions. If multiple constraints are needed, combine them into a constraint list (**ConstraintList**).

(4) Formulate optimization model

Create the optimization model (**Model**) from the objective function and constraints.

(5) Specify solver client attributes

Specify a machine or solver to use and create a solver client.

(6) Execute solver

Pass the optimization model and solver client, and run the optimization through the solve function. The result of the execution is returned as **Result**, and the objective function value and the variables of the optimal solution can be obtained.

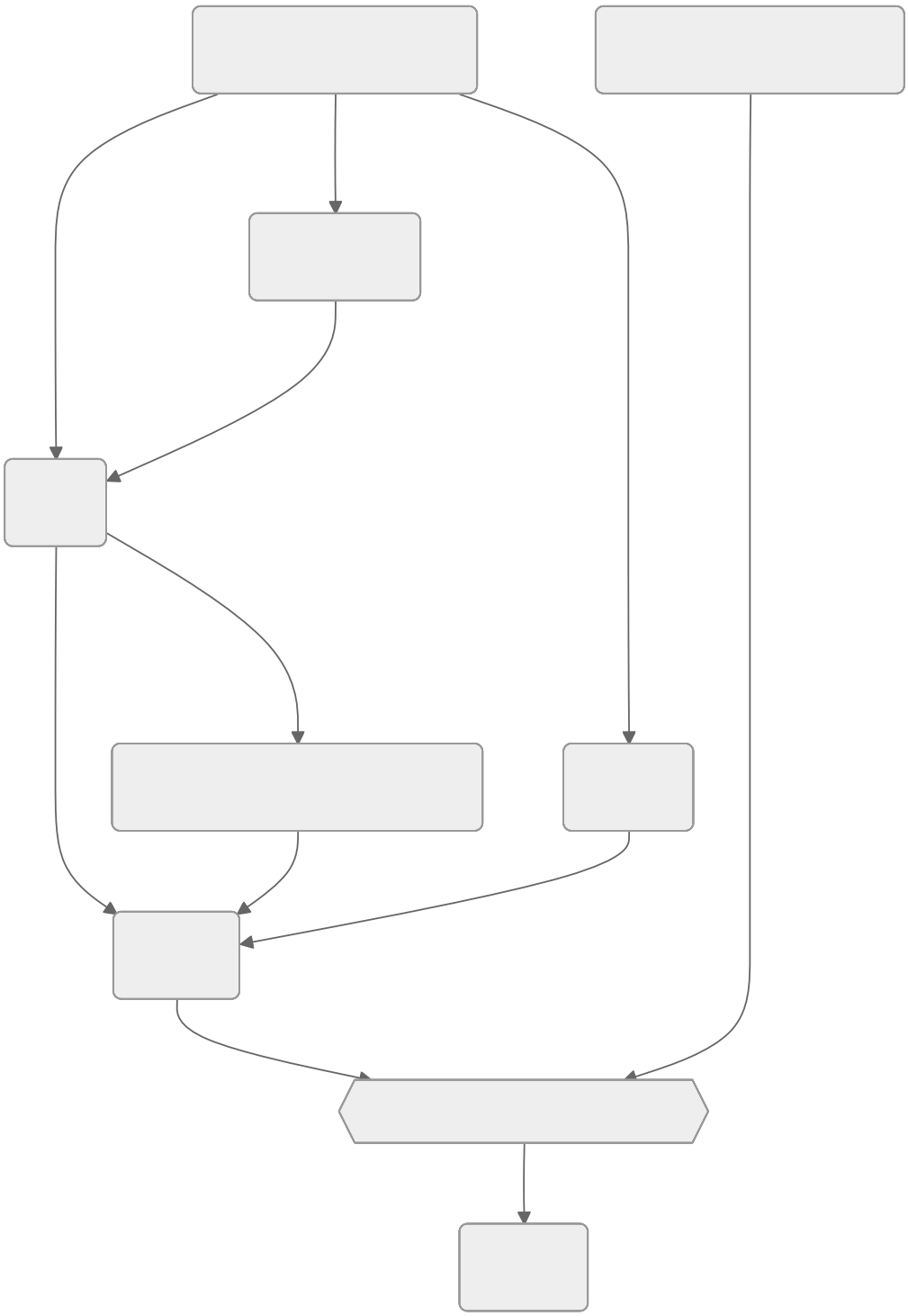

Figure 2: A schematic system architecture assumed in this specification

3 Decision Variables

The SDK MUST support the following classes and methods to determine the decision variables.

3.1 Variable Generator

The **VariableGenerator** class generates decision variables. This class MUST support the following methods.

Method	Return type	Description
scalar()	**Poly**	Create a single variable
array()	**PolyArray**	Create multiple variables

3.2 Methods and Arguments

The above methods MUST support the following types of decision variables as arguments.

Argument	Type	Variable type	Description
variable_type	string	"Binary"	A variable taking the value 0 or 1
		"Ising"	A variable taking the value -1 or 1
		"Integer"	A variable taking an integer value
		"Real"	A variable taking a real value

The above methods MUST also support the following arguments.

Argument	Type	Description
name	string	Specify the name of the variables
bounds	tuple(number, number)	[Integer or real variable only] Specify the range of possible values

3.2.1 scalar()

Define the **scalar()** method that creates a single variable. The variable MUST be returned as an instance of the **Poly** class, which represents the polynomial of the variable.

3.2.2 array()

Define the **array()** method that creates multiple variables at once. An array of variables (NumPy-like multidimensional array format) MUST be returned as an instance of the **PolyArray** class, which represents the polynomial array of the variable. This method MUST support the following argument to specify the shape of the array.

Argument	Type	Description
shape	int	tuple[int, . . .]

Once the new variables are created, the names MUST be automatically appended with a string representing the array index. The elements and slices MUST be retrievable in the same way as a Python list or NumPy array. The lower and upper bounds for variables MUST be specified collectively at the time of creation.

3.3 Get variable information

Define the **as_variable()** method that gets information about generated variables. The following information MUST be supported by this method.

Attribute	Type	Description
name	string	Variable name
id	int	Variable ID number An integer value assigned starting from 0 in order of issuance
type	**VariableType**	Variable type
lower_bound	float	[Integer or real variable only] Variable lower bound *None* means -∞
upper_bound	float	[Integer or real variable only] Variable upper bound *None* means +∞

The information about variables issued by the **VariableGenerator** class MUST be retrievable by using the **variables** attribute. The names, lower bounds, and upper bounds of variables MUST be modifiable later.

3.4 Example code

In the example code, the library name is represented as “SDK,” but you should replace it with the actual library name as appropriate. The same applies to the following example codes.

```python
from SDK import VariableGenerator

gen = VariableGenerator()
q = gen.scalar("Binary")                # print(q) -> q_0
i = gen.scalar("Ising", name="is")      # print(i) -> is
r = gen.scalar("Real", bounds=(2.0, 3.0))   # 2.0 <= r <= 3.0

q_arr = gen.array("Integer", 3, bounds=(1, 3))
"""
print(q_arr) -> [q_0, q_1, q_2]
print(q_arr[0]) -> q_0
print(q_arr[:2]) -> [q_0, q_1]
"""
q_mlt_arr = gen.array("Binary", shape=(2, 3))
"""
print(q_mlt_arr) ->
[[q_{0,0}, q_{0,1}, q_{0,2}],
   [q_{1,0}, q_{1,1}, q_{1,2}]]
print(q_mlt_arr[0, 0]) -> q_{0,0}
print(q_mlt_arr[0, :]) -> [q_{0,0}, q_{0,1}, q_{0,2}]
"""

# print(q.as_variable()) -> {name: q_0, id: 0, type: Binary}
vars = gen.variables
"""
print(vars) ->
[Variable({name: q_0, id: 0, type: Binary}),
 Variable({name: is, id: 1, type: Ising}),
 Variable({name: r_0, id: 2, type: Real, lower_bound: 2, upper_bound: 3})]
"""

vars[2].name = "r"
vars[2].lower_bound = 0.0
"""
print(vars[2]) ->
{name: r, id: 2, type: Real, lower_bound: 0, upper_bound: 3}
"""
```

4 Polynomials

The SDK MUST support the following classes and methods to determine the polynomials. The SDK MUST be able to use the polynomial created as an objective function of a combinatorial optimization problem without modification.

4.1 Arithmetic of polynomial

The SDK MUST support the creation of polynomial expressions by performing quadrature operations and exponentiation on variables created by **VariableGenerator**. It MUST be able to include variables from different arrays or of different types within the same polynomial, provided they were all created by the same **VariableGenerator**. The SDK MUST also support the following logical operators.

Operator	Effect
& (logical AND)	x & y is equivalent to x * y.
| (logical OR)	
^ (exclusive OR)	x ^ y is equivalent to -2 * x * y + x + y.

4.2 Arithmetic of polynomial arrays

The SDK MUST provide the **sum()** method that creates polynomial expressions from **PolyArray**. **PolyArray** MUST also be able to perform quadratic operations with numbers and NumPy arrays.

Method	Argument	Return type	Description
sum()	axis = None	int	tuple[int, ...]

4.3 Attributes and methods of polynomial

Polynomial class **Poly** MUST support the following methods and attributes to get and change information about polynomials.

Method / Attribute	Return type	Description
degree()	int	Degree of the polynomial
is_number() / is_linear() / is_quadratic()	bool	Whether the polynomial is below a certain degree
is_variable()	bool	Whether the polynomial can be considered a single variable i.e. a one-dimensional monomial with coefficient 1
variables	list[**Variables**]	Information about all variables in the polynomial
substitute()	**Poly**	The result of assigning numbers or other polynomials to variables of a polynomial expression

4.4 Example code

```
from SDK import VariableGenerator
import numpy as np

gen = VariableGenerator()
q = gen.array("Binary", 6)
p = -q[0] + 2.3 * q[1] * q[2] - (q[3] + q[4]) ** 2 * q[5]
# print(p) -> - 2 q_3 q_4 q_5 + 2.3 q_1 q_2 - q_3 q_5 - q_4 q_5 - q_0

q = gen.array("Binary", 3)
s = gen.array("Ising", 2)
n = gen.scalar("Integer", bounds=(-1, 2))
p = q[0] + s[1] - 2 * n
# print(p) -> q_0 + s_1 - 2 n_0

q = gen.array("Binary", shape=(3, 3))
"""
print(q.sum()) ->
   q_{0,0} + q_{0,1} + q_{0,2} + q_{1,0} + q_{1,1} + q_{1,2} + q_{2,0} + q_{2,1} + q_{2,2}
print(q.sum(axis=1)) ->
   [q_{0,0} + q_{0,1} + q_{0,2},
    q_{1,0} + q_{1,1} + q_{1,2},
    q_{2,0} + q_{2,1} + q_{2,2}]
print(q.sum(axis=0)) ->
   [q_{0,0} + q_{1,0} + q_{2,0},
 q_{0,1} + q_{1,1} + q_{2,1},
 q_{0,2} + q_{1,2} + q_{2,2}]
print(2 * q) ->
   [[2 q_{0,0}, 2 q_{0,1}, 2 q_{0,2}],
 [2 q_{1,0}, 2 q_{1,1}, 2 q_{1,2}],
 [2 q_{2,0}, 2 q_{2,1}, 2 q_{2,2}]]
"""

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
"""
print(q * a) ->
   [[  q_{0,0}, 2 q_{0,1}, 3 q_{0,2}],
 [4 q_{1,0}, 5 q_{1,1}, 6 q_{1,2}],
 [7 q_{2,0}, 8 q_{2,1}, 9 q_{2,2}]]
"""
q = gen.array("Binary", 4)
p = q[0] * q[1]
"""
p.degree() -> 2
p.is_number() -> False
p.is_linear() -> False
p.is_quadratic() -> True
```



```
q[0].is_variable() -> True
(q[0] + 1).is_variable() -> True
(2 * q[0]).is_variable() -> False

(q[0] + 2 * q[1]).variables ->
   [Variable({name: q_0, id: 0, type: Binary}),
 Variable({name: q_1, id: 1, type: Binary})]
"""
p = q[0] + q[1]
```

5 Constraints

The SDK MUST support setting constraints on the range of possible values for each variable, as well as constraints on the range of possible values for polynomial expressions. The SDK MUST support the following classes, methods, and operations for setting constraints on decision variables and for constructing constraint objects using polynomials.

5.1 Fix values

The SDK MUST support fixing the values of decision variables, by replacing part of the variable array with numeric values in advance. The SDK MUST also support imposing a constraint where a polynomial with another variable represents a variable by replacing part of the variable array with a polynomial.

5.2 Limit the range of variable values

See chapter Methods and Arguments

5.3 Limit the range of polynomial values

The SDK MUST manage equality, inequality, and other expressions representing constraints on the range of possible polynomial values as constraint objects of the **Constraint** class.

5.3.1 Equality constraints

The SDK MUST support the following helper functions to create constraint objects that represent equalities. The SDK MUST support assigning labels to these constraints.

Helper function	Arguments	Description
equal_to()	**Poly**	**PolyArray**, float
one_hot()	**Poly**	**PolyArray**

5.3.2 Inequality constraints

The SDK MUST support the following helper functions to create constraint objects representing inequalities. The SDK MUST support assigning labels to these constraints.

Helper function	Arguments	Description
less_equal()	**Poly**	**PolyArray**, float
greater_equal()	**Poly**	**PolyArray**, float
clamp()	**Poly**	**PolyArray**, tuple[float, float]

5.3.3 Constraint list

The SDK MUST support the handling of multiple constraints as a **ConstraintList**. The SDK MUST also support an empty **ConstraintList** object. A **ConstraintList** object MUST be generated by adding **Constraint** objects together. **ConstraintList** MUST also support adding a **Constraint** object using the + or += operator.

5.3.4 Constraint weights

Multiplying a **Constraint** object by a number MUST multiply its weight. Multiplication by numbers for **weight** MUST also be defined for **ConstraintList** objects. **Constraint** MUST support the **weight** attribute to obtain and set the weight of a constraint object. The default value of **weight** MUST be 1.

5.4 Example code

```
from SDK import VariableGenerator, equal_to, one_hot, less_equal, greater_equal, clamp

gen = VariableGenerator()
q = gen.array("Binary", shape=(3, 3))

q[0, :] = 0
q[:, 0] = 0
q[0, 0] = 1
"""
print(q) ->
   [[      1,       0,       0],
 [      0, q_{1,1}, q_{1,2}],
 [      0, q_{2,1}, q_{2,2}]]
"""
p = q.sum() # print(p) -> q_{1,1} + q_{1,2} + q_{2,1} + q_{2,2} + 1
q[2, 2] = q[1, 1]
"""
print(q) ->
   [[      1,       0,       0],
 [      0, q_{1,1}, q_{1,2}],
 [      0, q_{2,1}, q_{1,1}]]
"""

q = gen.array("Binary", shape=(3, 3))
c = equal_to(q[0, 0] + q[1, 1] + q[2, 2], 1)
# print(c) -> q_{0,0} + q_{1,1} + q_{2,2} == 1 (weight: 1)
c = equal_to(q[0, 0] + q[1, 1] + q[2, 2], 1, label="diagonal sum")
# print(c) -> diagonal sum: q_{0,0} + q_{1,1} + q_{2,2} == 1 (weight: 1)

c = equal_to(q[0], 1, label="1st row sum")
# print(c) -> 1st row sum: q_{0,0} + q_{0,1} + q_{0,2} == 1 (weight: 1)

c = one_hot(q[0], label="1st row one-hot")
# print(c) -> 1st row one-hot: q_{0,0} + q_{0,1} + q_{0,2} == 1 (weight: 1)

c_le = less_equal(q[0], 2)
c_ge = greater_equal(q[0], 2)
c_bw = clamp(q[0], (1, 2))
"""
print(c_le) -> q_{0,0} + q_{0,1} + q_{0,2} <= 2 (weight: 1)
print(c_ge) -> q_{0,0} + q_{0,1} + q_{0,2} >= 2 (weight: 1)
print(c_bw) -> 1 <= q_{0,0} + q_{0,1} + q_{0,2} <= 2 (weight: 1)
"""

q = gen.array("Binary", shape=(3, 3))
c0 = equal_to(q[0], 1)
c1 = equal_to(q[1], 1)
clist = c0 + c1
"""
print(clist) ->
   [q_{0,0} + q_{0,1} + q_{0,2} == 1 (weight: 1),
    q_{1,0} + q_{1,1} + q_{1,2} == 1 (weight: 1)]
"""
clist += equal_to(q[2], 1)
"""
print(clist) ->
```

6 Model

The SDK MUST support the following classes, methods, and operations to formulate the combinatorial optimization problem using decision variables, objective functions, and constraints.

6.1 Model class

The SDK MUST represent combinatorial optimization problems as instances of the **Model** class. The **Model** class constructor MUST accept **Poly** or/and **Constraint / ConstraintList** as arguments to represent the problem. **Model** objects MUST also be generated by adding **Poly** and **Constraint / ConstraintList**.

6.2 Model attributes

The **Model** class MUST provide the following attributes.

Attribute	Type	Description
objective	**Poly**	The objective function of the model
constraints	**ConstraintList**	Constraints of the model

6.3 Example code

```
from SDK import VariableGenerator, Model, equal_to, one_hot

gen = VariableGenerator()
q = gen.array("Binary", shape=(2, 3))

objective = q[0, 0] * q[0, 1] - q[0, 2]
constraint1 = equal_to(q[0, 0] + q[0, 1] - q[0, 2], 0)
constraint2 = one_hot(q[1, :])
constraint_list = constraint1 + constraint2

model = objective + constraint1
"""
print(model) ->
minimize:
     q_{0,0} q_{0,1} - q_{0,2}
subject to:
  q_{0,0} + q_{0,1} - q_{0,2} == 0 (weight: 1)
"""
model = objective + constraint_list
"""
print(model) ->
minimize:
     q_{0,0} q_{0,1} - q_{0,2}
subject to:
  q_{0,0} + q_{0,1} - q_{0,2} == 0 (weight: 1),
  q_{1,0} + q_{1,1} + q_{1,2} == 1 (weight: 1)
"""
model = Model(objective, constraint_list)
"""
print(model) ->
minimize:
     q_{0,0} q_{0,1} - q_{0,2}
subject to:
  q_{0,0} + q_{0,1} - q_{0,2} == 0 (weight: 1),
  q_{1,0} + q_{1,1} + q_{1,2} == 1 (weight: 1)
"""
model = Model(objective)
"""
print(model) ->
minimize:
  q_{0,0} q_{0,1} - q_{0,2}
"""
model = Model(constraint1)
"""
print(model) ->
minimize:
  0
subject to:
  q_{0,0} + q_{0,1} - q_{0,2} == 0 (weight: 1)
"""
model = Model(constraint_list)
"""
print(model) ->
minimize:
  0
subject to:
```

7 Solver Client

The SDK MUST provide a **SolverClient** class that abstracts each solver. **SolverClient** class MUST be able to set the attributes needed to meet the solver's specifications.
The following are some common attributes.

- connection point
- API token
- execution parameters

SolverClient class MUST also contain information about the solver's capabilities, such as the types of variables, constraints, degrees, and so on that it can handle.

```
from SDK import XXXClient
from datetime import timedelta

client = XXXClient()
client.token = "YOUR_API_TOKEN"
client.parameters.timeout = timedelta(milliseconds=1000)
"""
print(client) ->
{"url":"https://XXX.com","token":"YOUR_API_TOKEN",
"parameters":{"timeout":1000}}
"""
```

8 Solver Execution

The SDK MUST support the following classes and methods to solve a combinatorial optimization problem.

8.1 Solve function

The SDK MUST provide the **solve()** function to perform combinatorial optimization.

This function MUST take a **Model** object as its first argument and a solver client object as its second argument, optimizing the model using the solver associated with the solver client. The return value MUST be a **Result** object.

Argument	Type	Description
model	**Model**	Created in Model Formulation
client	**Client**	Created in Solver Client

8.2 Retrieving the result

A **Result** object returned by the solve function MUST contain information about the solution returned by the solver and the time taken to run it. **Result** MUST contain not only the best solution but also multiple solutions. If **Result** contains multiple solutions, the best solution MUST be provided as the **best** attribute. The best solution refers

to the solution that satisfies all the constraints and has the smallest objective function value among the multiple solutions.

Attribute	Type	Description
best	**Solution**	The best solution

Each solution MUST be obtained as an instance of the **Solution** class by indexing on **Result**. **Solution** class MUST support the following attributes.

Attribute	Type	Description
objective	float	The value of the objective function
values	**Values**	The value of each variable in the solution
feasible	bool	Whether the constraint is satisfied or not
time	timedelta	The time at which the solver finds the solution

8.3 Example code

```
from SDK import VariableGenerator, one_hot, XXXClient, solve
from datatime import timedelta

gen = VariableGenerator()
q = gen.array("Binary", 3)

objective = q[0] * q[1] - q[2]
constraint = one_hot(q)

model = objective + constraint

client = XXXClient()
client.token = "YOUR_API_TOKEN"
client.parameters.timeout = timedelta(milliseconds=1000)

result = solve(model, client)
"""
print(result.best.objective) -> -1.0
print(result.best.values) -> Values({Poly(q_0): 0, Poly(q_1): 0, Poly(q_2): 1})
print(result.best.feasible) -> True
print(result.best.time) -> datetime.timedelta(microseconds=27965)
"""
```

9 References

This chapter describes the definition and specification of the objects that SDK MUST support.

9.1 Algebraic classes and numerics

9.1.1 SDK.Poly class

Method	Return type
as_variable(self)	**Variable**
degree(self)	int
evaluate(self, values: Values)	float
is_linear(self)	bool
is_number(self)	bool
is_quadratic(self)	bool
is_variable(self)	bool

Attribute	Return type
id	int
lower_bound	float
name	string
type	**VariableType**
upper_bound	float
variables	list[**Variable**]

9.1.2 SDK.PolyArray class

Method	Return type
evaluate(self, values: Values)	ndarray[Any, dtype[numpy.float64]]
sum(self, axis: int	tuple[int, . . .]

Attribute	Return type
ndim	int
shape	tuple[int, . . .]
size	int

9.1.3 SDK.Variable class

Attribute	Return type
id	int
lower_bound	float
name	string
type	**VariableType**
upper_bound	float

9.1.4 SDK.VariableGenerator class

Method	Return type
array(self, type: str	**VariableType**, shape: tuple[int, ...]
scalar(self, type: str	**VariableType**, bounds: tuple[float

```
bounds : Defaults to (None, None)

name : Defaults to ''
```

Attribute	Return type
variables	list[**Variable**]

9.1.5 SDK.sum() function

Method	Return type
sum(list) sum(Iterator)	Any
sum(**PolyArray**, axis: int	Tuple[int, ...]

```
axis : Defaults to NoneConstraint classes and functions
```

9.1.6 SDK.VariableType enum class

Member	Type
Binary	**VariableType**.Binary
Integer	**VariableType**.Integer
Ising	**VariableType**.Ising
Real	**VariableType**.Real

9.2 Constraints

9.2.1 SDK.Constraint class

Method	Return type
is_satisfied(self, values: **Values**)	bool

Attribute	Return type
conditional	tuple[**Poly**, str, Union[float, tuple[float, float]]]
label	string
penalty	**Poly**
weight	float

9.2.2 SDK.ConstraintList class

Method	Return type
append(self, value: **Constraint**)	None
remove(self, value: **Constraint**)	None

9.2.3 Functions

Method	Arguments	Return type
equal_to() one_hot() less_equal() greater_equal()	poly: **Poly**, right: float, label: str = ”	**Constraint**
	array: **PolyArray**, right: float, label: str = ”, axis: int	tuple[int, . . .]
clamp()	poly: **Poly**, bounds: tuple[float	None, float
	array: **PolyArray**, bounds: tuple[float	None, float

9.3 Model classes and functions

9.3.1 SDK.Model class

Method	Return type
copy(self)	**Model**
get_variables(self)	list[**Variable**]

Attribute	Return type
constraints	**ConstraintList**
objective	**Poly**

Attribute	Return type
variables	list[**Variable**]

9.4 Solve (Solve classes and functions)

9.4.1 SDK.Result.Solution class

Attribute	Return type
feasible	bool
objective	float
time	timedelta
values	**Values**

9.4.2 SDK.Result class

Attribute	Return type
best	**Solution**
client_result	**Poly**
execution_time	list[**Variable**]
num_solves	int
response_time	timedelta
solutions	**SolutionList**
total_time	timedelta

9.4.3 SDK.Result.Values class

Method	Return type
items	View of list[tuple[**Poly**, float]]
keys	View of list[**Poly**]
values	View of list[float]