LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

TU Delft

Algorithmic QUBO Formulations

Algorithmic QUBO Formulations

- Most of the current QUBO formulations are arithmetic. E.g.:

$$H = A\sum_{v=1}^{n}\left(1-\sum_{j=1}^{N}x_{v,j}\right)^2 + A\sum_{j=1}^{n}\left(1-\sum_{v=1}^{N}x_{v,j}\right)^2 + A\sum_{(uv)\notin E}\sum_{j=1}^{N}x_{u,j}x_{v,j+1}$$

- Our idea of Algorithmic QUBOs:

Don‘t create QUBO formulations directly – create (parameterized) algorithms/functions which build QUBOs using element-wise addition instructions

Arithmetic QUBOs vs. Algorithmic QUBOs

$\mathcal{C} = \{C_1, C_2, \dots\}$ # set of constraints

QUBO matrix is a **weighted sum** of these constraints:

$$Q = A \cdot C_1 + \mathrm{B} \cdot C_2 + \cdots$$

$\mathcal{C} = \{C_1, C_2, \dots\}$ # set of constraints

QUBO matrix is built with an **algorithm** and **element-wise addition** instructions:

```
if … then:
    Q ← C_1
else:
    Q ← C_2
```

- Constraints C_i can also be parameterized
- C_i can also be whole QUBO formulations (or QUBO algorithms/functions) itself → reuse of preexisting QUBO formulations

Algorithmic QUBO for k-SAT

- Let C be a clause with k literals, there are 2^k possible assignments, of which $2^k - 1$ assignments satisfy the clause and 1 does not.

- Example: $(a \vee -b \vee c)$

a	b	c	SAT
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Algorithmic QUBO for k-SAT

- **Idea**:
  - all assignments that satisfy the clause → energy e
  - the one assignment that does not satisfy the clause → energy $e + 1$
  - the assignment which satisfy all clauses has the lowest energy

- **Our solution**:

1. We use $\mathrm{h} = \lceil \log_2(k+1) \rceil$ binary spins A_i to encode the number of literals $[0; k]$ which evaluate to True (given an assignment)

   Example: $(a \vee -b \vee c)$

Assignment (a,b,c)	#SAT
$(1{,}0{,}1)$	3
$(1{,}1{,}0)$	1
$(0{,}1{,}0)$	0

Algorithmic QUBO for k-SAT

Our solution:

1. We use $\mathrm{h} = \lceil \log_2(k+1) \rceil$ binary spins A_i to encode the number of literals $[0; k]$ which evaluated to True (given an assignment x)

   Example: $(a \vee -b \vee c)$

Assignment (a,b,c)	#SAT
$(1,0,1)$	3
$(1,1,0)$	1
$(0,1,0)$	0

$n(C)$ = # negative literals (in the example: 1)

$x(l)$ = assignment of the variable corresponding to literal l

$sign(l)$ = sign of the literal (in the example: $sign(a) = 1, sign(-b) = -1$)

$$n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) = \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j)$$

Algorithmic QUBO for k-SAT

- **Our solution**:

  1. We use $\mathrm{h} = [\log_2(k+1)]$ binary spins A_i to encode the number of literals $[0; k]$ which evaluate to True (given an assignment x)

Rewrite this equation as an optimization problem:

$$n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) = \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j)$$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \right)^2 \qquad (6)$$

Algorithmic QUBO for k-SAT

- **Our solution**:

1. We use $\mathrm{h} = \lceil \log_2(\mathrm{k}+1) \rceil$ binary spins $\mathrm{A_i}$ to encode the number of literals $[0; \mathrm{k}]$ which evaluate to True (given an assignment x)

2. $\#\mathrm{SAT} > 0 \quad \leftrightarrow \quad \exists\, \mathrm{A_i}\colon \mathrm{A_i} = 1 \quad \leftrightarrow \quad \mathrm{newC} = [\mathrm{A_1} \vee \cdots \vee \mathrm{A_h}]$

3. We continue recursively (anchor of the recursion is 2-SAT and 3-SAT, for which we use the well-known QUBO formulations).

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula*: f)
  init empty QUBO matrix Q;
  for $C \in f$ **do**
    `implementClause`(Q, C);
  end
  return Q;

`implementClause`(*QUBO matrix*: Q, *Clause*: C)
  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1 , \ \ldots \ , A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
    `implementClause`($Q, newC$);
  end
  return Q;

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula*: f)
  init empty QUBO matrix Q;
  for $C \in f$ **do**
    `implementClause`(Q, C);
  end
  return Q;

`implementClause`(*QUBO matrix*: Q, *Clause*: C)
  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1 , \ \dots \ , A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
    `implementClause`($Q, newC$);
  end
  return Q;

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)

init empty QUBO matrix Q;
for $C \in f$ **do**
  `implementClause`(Q, C);
end
return Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)

if $len(C) = 2$ **then**
  $Q \leftarrow OR(C)$;
else if $len(C) = 3$ **then**
  $X \leftarrow [A_1]$;
  $Q \leftarrow$ 3-SAT(C, A_1);
else
  $h = \lceil \log_2 (len(C) + 1) \rceil$;
  $X \leftarrow [A_1 , \ \ldots \ , A_h]$;
  $Q \leftarrow$ formula (6);
  $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
  `implementClause`$(Q, newC)$;
end
return Q;

Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula:* f)

  init empty QUBO matrix Q;
  for $C \in f$ **do**
    `implementClause`(Q, C);
  end
  return Q;

`implementClause`(*QUBO matrix:* Q, *Clause:* C)

  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1 , \ \ldots \ , A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
    `implementClause`($Q, newC$);
  end
  return Q;

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \right)^2 \tag{6}$$

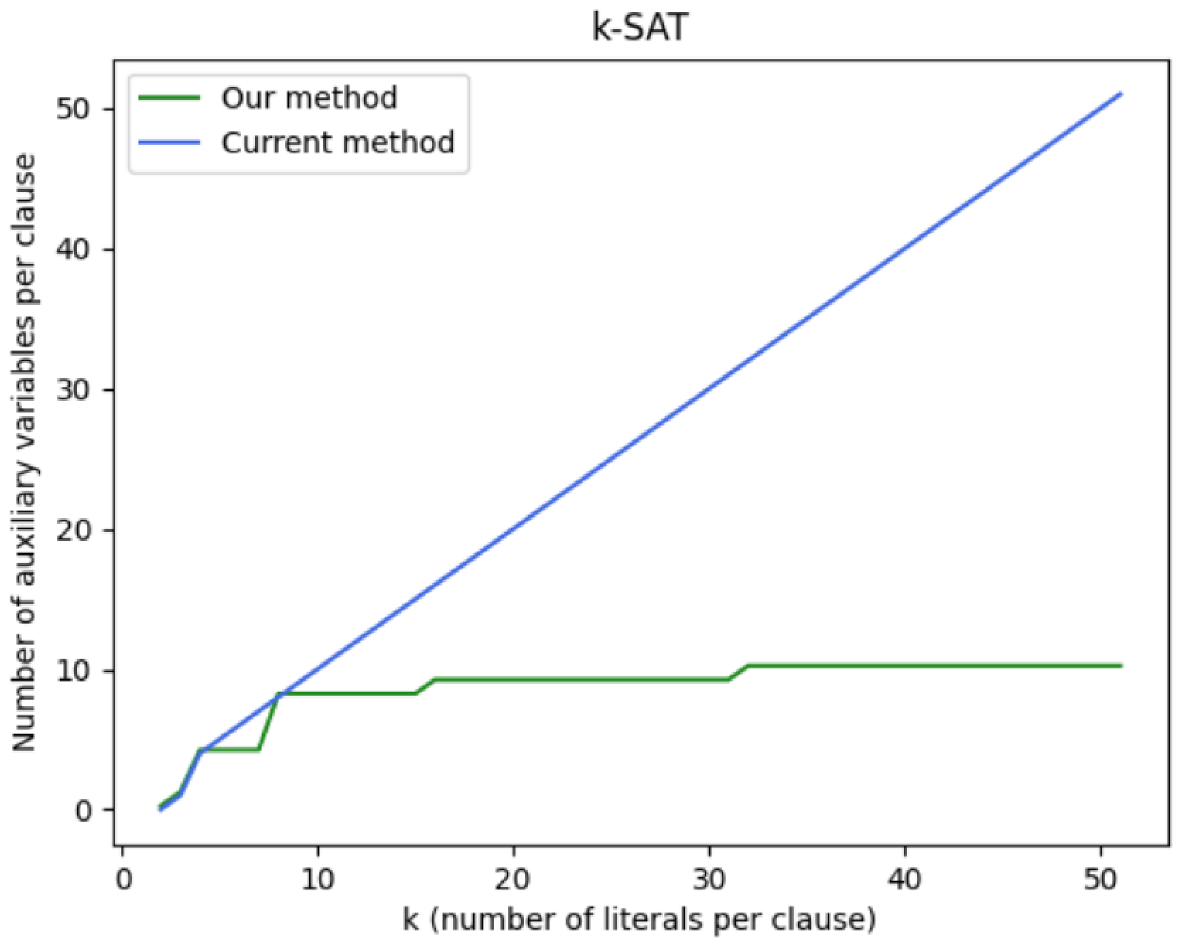
Algorithmic QUBO for k-SAT

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

Algorithmic QUBO for k-SAT



Algorithmic QUBO Formulations

Advantages of Algorithmic QUBOs:

1. more flexible

   Use branching and loops to decide which constraints are inserted (with which parameters) into the QUBO

2. easier to read/understand than arithmetic QUBOs (*imagine how many indices you would need to write the recursion as an arithmetic QUBO*)

Example

$$C = (a \vee -b \vee c \vee d \vee -e) \wedge$$
$$(-a \vee b \vee -c \vee d \vee e)$$

$$C = (a \lor -b \lor c \lor d \lor -e) \land$$
$$(-a \lor b \lor -c \lor d \lor e)$$

List of clauses:

$(a \lor -b \lor c \lor d \lor -e)$

$(-a \lor b \lor -c \lor d \lor e)$

$$C = (a \vee -b \vee c \vee d \vee -e) \wedge (-a \vee b \vee -c \vee d \vee e)$$

List of clauses:

$$(a \vee -b \vee c \vee d \vee -e)$$

$$(-a \vee b \vee -c \vee d \vee e)$$

Algorithm 1: QUBO algorithm for (Max) k-SAT

fillQ(*Formula*: f)
- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - implementClause(Q, C);
- **end**
- **return** Q;

implementClause(*QUBO matrix*: Q, *Clause*: C)
- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1, \ldots, A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \vee .. \vee A_h]$;
  - implementClause($Q, newC$);
- **end**
- **return** Q;

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A1];
        Q ← 3-SAT(C, A1);
    else
        h = ⌈log2 (len(C) + 1)⌉;
        X ← [A1 , ... , Ah];
        Q ← formula (6);
        newC = [A1 ∨ .. ∨ Ah];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - `implementClause`(Q, C);
- **end**
- **return** Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1 , \ \ldots \ , A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
  - `implementClause`$(Q, newC)$;
- **end**
- **return** Q;

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - `implementClause`(Q, C);
- **end**
- **return** Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1, \ldots, A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \lor \ldots \lor A_h]$;
  - `implementClause`$(Q, newC)$;
- **end**
- **return** Q;

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula:* f)
  init empty QUBO matrix Q;
  for $C \in f$ **do**
    `implementClause`(Q, C);
  end
  return Q;

`implementClause`(*QUBO matrix:* Q, *Clause:* C)
  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1, \dots, A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \lor \dots \lor A_h]$;
    `implementClause`$(Q, newC)$;
  end
  return Q;

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

$$h = \lceil \log_2(6+1) \rceil = 3$$

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

fillQ(*Formula: f*)
  init empty QUBO matrix Q;
  for $C \in f$ **do**
    implementClause(Q, C);
  end
  return Q;

implementClause(*QUBO matrix: Q, Clause: C*)
  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1, \ \ldots \ , A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \lor \ .. \ \lor A_h]$;
    implementClause($Q, newC$);
  end
  return Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a									
b									
c									
d									
e									
f									
A1									
A2									
A3									

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(-a \vee b \vee -c \vee d \vee e \vee f)$

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a									
b									
c									
d									
e									
f									
A1									
A2									
A3									

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\left(A_j\right) \right)^2 \qquad (6)$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a									
b									
c									
d									
e									
f									
A1									
A2									
A3									

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\left(A_j\right)\right)^2 \qquad (6)$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

$(a \vee -b \vee c \vee d \vee -e \vee f)$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \right)^2 \qquad (6)$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

$(a \vee -b \vee c \vee d \vee -e \vee f)$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j)\right)^2 \qquad (6)$$

$$(2 + a - b + c + d - e + f - A_1 - 2A_2 - 4A_3)^2$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

$(a \vee -b \vee c \vee d \vee -e \vee f)$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j) \right)^2 \qquad (6)$$

$$(2 + a - b + c + d - e + f - A_1 - 2A_2 - 4A_3)^2$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

$(a \vee -b \vee c \vee d \vee -e \vee f)$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\,(A_j)\right)^2 \qquad (6)$$

$$(2 + a - b + c + d - e + f - A_1 - 2A_2 - 4A_3)^2$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

$$(a \vee -b \vee c \vee d \vee -e \vee f)$$

$$\left(n(C) + \sum_{l \in C} sign\,(l) \cdot x\,(l) - \sum_{j=1}^{h} 2^{j-1} \cdot x\left(A_j\right) \right)^2 \qquad (6)$$

$$(2 + a - b + c + d - e + f - A_1 - 2A_2 - 4A_3)^2$$

$n(C)$ = # negative literals
$x(l)$ = assignment of the variable corresponding to literal l
$sign(l)$ = sign of the literal

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
 init empty QUBO matrix Q;
 for $C \in f$ **do**
  `implementClause`(Q, C);
 end
 return Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
 if $len(C) = 2$ **then**
  $Q \leftarrow OR(C)$;
 else if $len(C) = 3$ **then**
  $X \leftarrow [A_1]$;
  $Q \leftarrow \text{3-SAT}(C, A_1)$;
 else
  $h = \lceil \log_2 (len(C) + 1) \rceil$;
  $X \leftarrow [A_1, \ \dots \ , A_h]$;
  $Q \leftarrow$ formula (6);
  $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
  `implementClause`$(Q, newC)$;
 end
 return Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
    init empty QUBO matrix Q;
    for $C \in f$ **do**
        `implementClause`(Q, C);
    end
    return Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
    if $len(C) = 2$ **then**
        $Q \leftarrow OR(C)$;
    else if $len(C) = 3$ **then**
        $X \leftarrow [A_1]$;
        $Q \leftarrow$ 3-SAT(C, A_1);
    else
        $h = \lceil \log_2 (len(C) + 1) \rceil$;
        $X \leftarrow [A_1, \ \dots \ , A_h]$;
        $Q \leftarrow$ formula (6);
        $newC = [A_1 \lor \ .. \ \lor A_h]$;
        `implementClause`$(Q, newC)$;
    end
    return Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$
3. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - `implementClause`(Q, C);
- **end**
- **return** Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1, \dots, A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \vee .. \vee A_h]$;
  - `implementClause`$(Q, newC)$;
- **end**
- **return** Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$
3. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(A1 \lor A2 \lor A3)$
3. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
    init empty QUBO matrix Q;
    for $C \in f$ **do**
        `implementClause`(Q, C);
    end
    return Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
    if $len(C) = 2$ **then**
        $Q \leftarrow OR(C)$;
    else if $len(C) = 3$ **then**
        $X \leftarrow [A_1]$;
        $Q \leftarrow$ 3-SAT(C, A_1);
    else
        $h = \lceil \log_2 (len(C) + 1) \rceil$;
        $X \leftarrow [A_1, \ldots, A_h]$;
        $Q \leftarrow$ formula (6);
        $newC = [A_1 \lor \ .. \ \lor A_h]$;
        `implementClause`($Q, newC$);
    end
    return Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(A1 \lor A2 \lor A3)$
3. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)

- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - `implementClause`(Q, C);
- **end**
- **return** Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)

- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1, \dots, A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \lor .. \lor A_h]$;
  - `implementClause`$(Q, newC)$;
- **end**
- **return** Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3
a	5	-2	2	2	-2	2	-2	-4	-8
b		-3	-2	-2	2	-2	2	4	8
c			5	2	-2	2	-2	-4	-8
d				5	-2	2	-2	-4	-8
e					-3	-2	2	4	8
f						5	-2	-4	-8
A1							-3	4	8
A2								-4	16
A3									0

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$
3. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula: f*)
  init empty QUBO matrix Q;
  for $C \in f$ **do**
    `implementClause`(Q, C);
  end
  return Q;

`implementClause`(*QUBO matrix: Q, Clause: C*)
  if $len(C) = 2$ **then**
    $Q \leftarrow OR(C)$;
  else if $len(C) = 3$ **then**
    $X \leftarrow [A_1]$;
    $Q \leftarrow$ 3-SAT(C, A_1);
  else
    $h = \lceil \log_2 (len(C) + 1) \rceil$;
    $X \leftarrow [A_1, \ ..., \ A_h]$;
    $Q \leftarrow$ formula (6);
    $newC = [A_1 \ \vee \ .. \ \vee \ A_h]$;
    `implementClause`$(Q, newC)$;
  end
  return Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	4	8	
A2								-4	16	
A3									0	
A4										

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$
3. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$
3. $(-a \vee b \vee -c \vee d \vee e \vee f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

`fillQ`(*Formula:* f)
- init empty QUBO matrix Q;
- **for** $C \in f$ **do**
  - `implementClause`(Q, C);
- **end**
- **return** Q;

`implementClause`(*QUBO matrix:* Q, *Clause:* C)
- **if** $len(C) = 2$ **then**
  - $Q \leftarrow OR(C)$;
- **else if** $len(C) = 3$ **then**
  - $X \leftarrow [A_1]$;
  - $Q \leftarrow$ 3-SAT(C, A_1);
- **else**
  - $h = \lceil \log_2 (len(C) + 1) \rceil$;
  - $X \leftarrow [A_1, \ \ldots \ , A_h]$;
  - $Q \leftarrow$ formula (6);
  - $newC = [A_1 \ \vee \ .. \ \vee A_h]$;
  - `implementClause`$(Q, newC)$;
- **end**
- **return** Q;

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

6 ancilla qubits $\rightarrow$ 4 ancilla qubits

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(A1 \lor A2 \lor A3)$
3. $(-a \lor b \lor -c \lor d \lor e \lor f)$

Algorithm 1: QUBO algorithm for (Max) k-SAT

```
fillQ(Formula: f)
    init empty QUBO matrix Q;
    for C ∈ f do
        implementClause(Q, C);
    end
    return Q;

implementClause(QUBO matrix: Q, Clause: C)
    if len(C) = 2 then
        Q ← OR(C);
    else if len(C) = 3 then
        X ← [A_1];
        Q ← 3-SAT(C, A_1);
    else
        h = ⌈log_2 (len(C) + 1)⌉;
        X ← [A_1 , ... , A_h];
        Q ← formula (6);
        newC = [A_1 ∨ .. ∨ A_h];
        implementClause(Q, newC);
    end
    return Q;
```

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

Let's get into the code ...

Let's get into the code ... after the break

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

$[1, 0, 1, 1, 0, 1]$

↓

$E = -5$

List of clauses:

1. $(a \lor -b \lor c \lor d \lor -e \lor f)$
2. $(A1 \lor A2 \lor A3)$

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

$[1, 0, 1, 1, 0, 1]$

$\downarrow$

$E = -5$

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

$$[0,0,0,0,0,1]$$

$$\downarrow$$

$$E = -5$$

List of clauses:

1. $(a \vee -b \vee c \vee d \vee -e \vee f)$
2. $(A1 \vee A2 \vee A3)$

QUBO Matrix Q:

	a	b	c	d	e	f	A1	A2	A3	A4
a	5	-2	2	2	-2	2	-2	-4	-8	
b		-3	-2	-2	2	-2	2	4	8	
c			5	2	-2	2	-2	-4	-8	
d				5	-2	2	-2	-4	-8	
e					-3	-2	2	4	8	
f						5	-2	-4	-8	
A1							-3	6	8	-2
A2								-4	16	-2
A3									-1	1
A4										1

$[0, 1, 0, 0, 1, 0]$

$\downarrow$

$E = -4$