SAT: Modelling

Mohamed Siala
https://siala.github.io

INSA-Toulouse & LAAS-CNRS

January 16, 2022

Modelling in SAT: The example of Graph Coloring

Modelling in SAT: The example of Graph Coloring

- Propose a SAT model for this problem
  (hint $x \rightarrow y$ is equivalent to $\neg y \rightarrow x$ and both are translated into the clause $\neg x \vee y$).

Modelling in SAT: The example of Graph Coloring

- Propose a SAT model for this problem
  (hint $x \rightarrow y$ is equivalent to $\neg y \rightarrow x$ and both are translated into the clause $\neg x \vee y$).

The Example of Graph Coloring: A Possible Model

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \dots x_i^k$$

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \dots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \dots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

  (This is a translation of $x_i^a \rightarrow \neg x_i^b$)

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

- Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \dots x_i^k$$

- If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

  (This is a translation of $x_i^a \rightarrow \neg x_i^b$)

- Forbid two nodes that share an edge to be coloured with the same colour

$$\forall \{i, j\} \in E, \forall a \in [1, k] : \neg x_i^a \rightarrow \neg x_j^a$$

  (This is a translation of $x_i^a \rightarrow \neg x_j^a$)

The Example of Graph Coloring: The Model Size

What is the (space) size of the model?

- $n \times k$ Boolean variables
- Constraints form 1: n clauses with k literals each
- Constraints form 2: $n \times k^2$ binary clauses
- Constraints form 3: $m \times k$ binary clauses

The Example of Graph Coloring: The Minimization Version

- Propose a method that uses SAT for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

A Straightforward Approach

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:
  - **Decreasing linear Search:** Run iteratively $SAT(V, E, UB - 1), SAT(V, E, UB - 2), \ldots$ until the problem is unsatisfiable. The last satisfiable value of k is the optimal value

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:
  - **Decreasing linear Search:** Run iteratively $SAT(V, E, UB-1), SAT(V, E, UB-2), \ldots$ until the problem is unsatisfiable. The last satisfiable value of k is the optimal value
  - **Binary search:** Run iteratively $SAT(V, E, z)$ as long as $UB > LB$ where $z = \lceil (UB - LB)/2 \rceil$. If the result is satisfiable, then and $UB \leftarrow z$ otherwise $LB \leftarrow z$

Upper/Lower Bound?

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
    - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex
  - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the the upper bound is the number of different colours used.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
    - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex
    - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
    - The resulting colouring is valid and the the upper bound is the number of different colours used.
    - The run time complexity is $O(n^2 \times m)$
- Lower bound: Well, we can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
    - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \dots n\}$ for each vertex
    - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
    - The resulting colouring is valid and the the upper bound is the number of different colours used.
    - The run time complexity is $O(n^2 \times m)$
- Lower bound: Well, we can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$
- Lower bound: Well, we can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.
- An alternative approach is to look for valid theoretical bounds in the literature.

Modelling Cardinality Constraints

- The general form of cardinality constraints is the following:

$$a \leq \sum_{1}^{n} x_i \leq b$$

  where a and b are positive integers and $x_1 \dots x_n$ are Boolean variables
- Cardinality constraints are everywhere!
- Many ways to encode such constraints. See for instance `https://www.carstensinz.de/papers/CP-2005.pdf`

Quadratic encoding for $\sum_1^n x_i = 1$

- At least one constraint:

$$x_1 \vee x_2 \ldots x_n$$

- at most one constraints:

$$\forall i, j : \neg x_i \vee \neg x_j$$

This generates one clause of size n and (n^2) binary clauses without introducing additional variables.

Linear encoding for $\sum_1^n x_i = 1$

New variables are added as follows: for $i \in [1, n], y_i$ is a new variable that is true iff $\sum_{l=1}^{l=i} x_l = 1$.

$$x_1 \vee x_2 \dots x_n$$

$$y_n^1$$

$$\forall i \in [1, n-1] : y_i \rightarrow y_{i+1}$$

$$\forall i \in [1, n-1] : y_i \rightarrow \neg x_{i+1}$$

$$\forall i \in [1, n] : x_i \rightarrow y_i$$

Size: n new variables, 1 $n-$ary clause and $3 \times n$ binary clauses,

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0,n] : y_i^0 \leftarrow 1$$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0,n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \ and \ \forall z \in [2,k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0, n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \; and \; \forall z \in [2, k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

$$\forall i \in [1, n], \forall z \in [1, k-1] : y_i^{z+1} \rightarrow y_i^z$$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0,n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \; and \; \forall z \in [2,k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

$$\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$$

$$\forall i \in [1,n-1], \forall z \in [1,k] : y_i^z \rightarrow y_{i+1}^z$$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0, k], \forall i \in [1, n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0, n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \ and \ \forall z \in [2, k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

$$\forall i \in [1, n], \forall z \in [1, k-1] : y_i^{z+1} \rightarrow y_i^z$$

$$\forall i \in [1, n-1], \forall z \in [1, k] : y_i^z \rightarrow y_{i+1}^z$$

$$\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$$

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0,n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \; and \; \forall z \in [2,k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

$$\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$$

$$\forall i \in [1,n-1], \forall z \in [1,k] : y_i^z \rightarrow y_{i+1}^z$$

$$\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$$

$$y_{i-1}^z \wedge x_i \rightarrow y_i^{z+1}$$

Linear encoding for $\sum_1^n x_i \geq k$

Size of the encoding:

- $\Theta(n \times k)$ variables
- $\Theta(n + k)$ unary clauses
- $\Theta(n \times k)$ binary clauses
- $\Theta(n \times k)$ ternary clauses

Linear encoding for $\sum_1^n x_i = k$?

Linear encoding for $\sum_1^n x_i = k$?

Linear encoding for $\sum_1^n x_i = k$?

- Encode $\sum_1^n x_i \geq k + 1$

Linear encoding for $\sum_1^n x_i = k$?

- Encode $\sum_1^n x_i \geq k+1$
- Force y_n^{k+1} to be false and y_n^k to be true

Size of the encoding: Same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $\sum_1^n x_i \leq k$?

Linear encoding for $\sum_1^n x_i \leq k$?

- Encode $\sum_1^n x_i \geq k+1$

Linear encoding for $\sum_1^n x_i \leq k$?

- Encode $\sum_1^n x_i \geq k+1$
- Force y_n^{k+1} to be false

Size of the encoding: Same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

- Encode $\sum_1^n x_i \leq b$

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

- Encode $\sum_1^n x_i \leq b$
- Force y_n^a to be true

Size of the encoding: Same as $\sum_1^n x_i \geq b$ (asymptotically)

Extensions: MaxSAT

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight
- Objective: satisfy hard clauses and maximizes the weighted sum of satisfied soft clauses.

Extensions: MaxSAT

- MaxSAT is an optimisation extension of SAT where some clauses are "hard" (must be satisfied) and others are "soft" (can be violated).
- The task is to find an assignment of the variables that satisfy the hard clauses and maximises the number of "soft" clauses
- MaxSAT:
  - Variables: Booleans, Clauses: hard and soft clauses
  - Maximisation problem: Is there an assignment of the variables that satisfy all the hard clauses, and maximises the number of satisfied soft clauses?
- Weighted MaxSAT: Extension of MaxSAT where every soft clause is associated with a weight
- Objective: satisfy hard clauses and maximizes the weighted sum of satisfied soft clauses.
- Check the MaxSAT competition

Example of applications for MaxSAT

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. In the (decision version of the) graph colouring problem, we are given k colours to colour the graph such that no two adjacent nodes share the same colour.

- Propose a MaxSAT model for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

Example of applications for MaxSAT

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. In the (decision version of the) graph colouring problem, we are given k colours to colour the graph such that no two adjacent nodes share the same colour.

- Propose a MaxSAT model for the minimisation version of the problem. That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.

The Example of Graph Coloring: A Possible MaxSAT Model

- We shall extend the previous model:
- Consider the previous model $SAT(V, E, k)$ with k an upper bound.
- All the previous clauses are hard.
- Add the following hard clauses:

$$\forall i \in [1, n], \forall a \in [1, k] : \neg u_a \to \neg x_i^a$$

- Eventually we can add symmetry constraints: $u_a \to u_{a-1}$
- Then add the soft clauses:

$$\forall a \in [1, k] : \neg u_a$$

- A MaxSAT Optimal solution satisfies all the hard coloring clauses (valid colouring) and maximizes the number of non used colours.

Extensions: Quantified Boolean Formula (QBF)

- A QBF has the form Q.F , where F is a CNF-SAT formulae, and Q is a sequence of quantified variables ($\forall x$ or $\exists x$).
- Example $\forall x, \exists y, \exists z,$ $(x \vee \neg y) \wedge (\neg y \vee z)$
- QBF Solver Competition: `https://www.qbflib.org/solvers_list.php`
- QBF is less used in practice

Other Extensions

- Satisfiability Modulo Theories
- Answer Set Programming
- More generally: Automated reasoning community
- Check the SAT/SMT summer schools
  `http://satassociation.org/sat-smt-school.html`