Reflections Vol. 1 (2013) 1–100

2-SAT Algorithm For Complete Set of Solutions

Wolfgang Scherer Wolfgang.Scherer@gmx.de

Abstract

The satoku matrix is constructed based on an inverted adjacency matrix as a data structure representing all possible solutions to a boolean satisfiability problem (k-SAT) as partial assignments with space requirements of $\mathcal{O}((m \cdot k)^2)$.

A computational algorithm (requirements update algorithm) is presented, which propagates the known consequences of conflicts throughout the satoku matrix in polynomial time.

It is shown that satisfiability of 2-SAT problems is determined after the first run of the requirements update algorithm.

A method is presented to generate a set of reliable assignments from the satoku matrix, which represent all solutions to the underlying 2-SAT problem.

The set of reliable assignments can be made unique by adding a conflict bias to the original CNF problem.

KEYWORDS: *boolean satisfiability, 2-SAT, SAT-solver, selection problem, satoku*

Contents

1. Preface

The description of the satoku matrix presents a peculiar satisfiability problem in itself. While trying to satisfy the expectation of established mathematical conventions, I found it quite hard to choose appropriate terms without being misleading.

On one hand it is necessary to use established terms to facilitate categorization, on the other hand it is obviously necessary to emphasize the differences to avoid inapplicable expectations from established preconceptions.

Although there is nothing revolutionary new in this algorithm, the deviation from convential interpretation of logic and its consequences leads to some interesting insights, but on the other hand it requires that the meaning of mathematically *not necessary* must be properly distinguished from the meaning of *irrelevant*. In my own experience I also found it to be important to avoid the fallacy, that since something is *true*, it should follow that it is *relevant*.

Therefore, some concepts of the algorithm are mentioned explicitely with a short rationale, to give a hint regarding the difference in perspective from established notions.

The algorithm removes special meaning from literals in clauses by abstracting them to generic vertices of a graph[1]. Besides the necessary structural constraints of conflicting literals as edges, the algorithm also **preserves the structural constraints of clauses**. The resulting class of **propositional selection problems** P has a narrower definition than the class of general selection problems S:

$$\forall f : f \in P \rightarrow f \in S,$$
$$\not\forall g : g \in S \rightarrow g \in P.$$

Note, that this does not mean, that some selection problems cannot be solved by mapping them to a satisfiability problem. It only means, that the graph of a general selection problem and the corresponding graph of a propositional selection problem are not necessarily isomorphic.

The algorithm further widens the narrow definition of problems in conjunctive normal form CNF to a **more general conjunction of clauses in disjunctive normal form DNF**, e.g.:

$$\begin{aligned}&(p \vee q) \wedge \\ &(r \vee s) \\ &\Leftrightarrow \\ &((p \wedge \mathtt{T}) \vee (q \wedge \mathtt{T})) \wedge \\ &((r \wedge \mathtt{T}) \vee (s \wedge \mathtt{T})),\end{aligned}$$

1. This often raises expectations of a graph algorithm. However, the graph representation of a selection problem is just the correct mapping for propositional selection problems. Only the scope of general graph theory is too wide to adequately handle selection problems represented by propositional formulas. The loss of information by ignoring (unnecessary, but not irrelevant) clauses makes propositional selection problems unnecessarily complex.

which allows to explore the usefulness of simple conflict maximization based on the logical equivalence of:

$$(p \vee q \vee r) = ((p) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q \wedge r)) \ .$$

See appendix B.2 for further details.

The requirement, that a variable and its negation must not appear in the same clause (redundancy removal) is dropped in favor of the convenient possibility to **represent a variable p as a clause**:

$$p \mapsto (p \vee \neg p).$$

See appendix B.1 for further details.

The term *conflict* is used in its general sense as a conflict between literals or partial assignments. There is **no relation to the term *conflict* of conflict driven clause learning CDCL**.

2. SAT Problems

In this article a general SAT problem P is a conjunction of one or more disjunctive clauses C_i, $i = (0, 1, \ldots, |P| - 1)$, consisting of one or more conjunctions A_{i_j}, $j = (0, 1, \ldots, |C_i| - 1)$ of one or more literals l_f, $f = (0, 1, \ldots, |A_{i_j}| - 1)$:

$$P = \bigwedge_{i=0}^{|P|-1} C_i, C_i = \bigvee_{j=0}^{|C_i|-1} A_{i_j}, A_{i_j} = \bigwedge_{f=0}^{|A_{i_j}|-1} l_f$$

A k-SAT problem consists entirely of clauses with the same size $k = |C_i|$, where $k = (1, 2, 3, \ldots)$:

$$P = \bigwedge_{i=0}^{|P|-1} \bigvee_{j=0}^{k-1} \bigwedge_{f=0}^{|A_{i_j}|-1} l_f$$

3. Independent Set Problem and Adjacency Matrix

As shown in David M. Mount, CMSC 451 Lecture Notes, Fall 2012, pp. 92, a 3-SAT problem can be mapped to an independent set problem in polynomial time.

The mapping to an independent set problem ensures, that only one literal can be selected from each clause.

Independent set problems can further be mapped to an adjacency matrix, which can be inverted directly during mapping by exchanging 0 and 1.

4. Satoku Matrix

The inverted adjacency matrix of the independent set graph ignores the clause restrictions, which were explicitly introduced during the mapping (and are therefore implicit in the final inverted adjacency matrix). However, the information is readily available during mapping and allows to subdivide the matrix into regions, which show the relation of the matrix to the mapped clauses.

This augmented inverted adjacency matrix is called satoku matrix (SM) to clearly indicate, that the algorithm is not necessarily related to graph theory and the independent set problem. The basic transformations of the satoku matrix preserve the properties of the initial matrix, therefore the results of transformations are consistent with graph theory. However, the consequences are outside the scope of this discussion.

Note: The satoku matrix was originally designed as optimal data structure to capture all information provided by a propositional problem in CNF for properly representing partial assignments for all possible solutions. The fact that it can also be interpreted as inverted adjacency matrix was detected much later and just proves that the mapping is correct. Using the correctness of the mapping to an inverted adjacency matrix as starting point just spares the reader the 100 odd pages of an inductive correctness proof based on full truth tables and the mutual exclusion relation. It also saves the author the need to find anybody to read said proof. But it requires to emphasize that the correlation to graph theory is a mere coincidence (although it could have been expected).

The properties of a SM are shown by mapping the following SAT-Problem (see appendix C for detailed construction):

$$
\begin{array}{l}
(\neg a \vee \neg b \vee \ \ c) \wedge \\
(\neg a \vee \ \ b \vee \neg c) \wedge \\
(\neg a \vee \ \ b \vee \ \ c) \wedge \\
(\ a \vee \neg b \vee \neg c) \wedge \\
(\ a \vee \neg b \vee \ \ c) \wedge \\
(\ a \vee \ \ b \vee \neg c) \wedge \\
(\ a \vee \ \ b \vee \ \ c) \wedge \\
(\ a \vee \neg a) \wedge \\
(\ b \vee \neg b) \wedge \\
(\ c \vee \neg c)
\end{array}
$$

The full satoku matrix for this problem looks like this:

P	− − −	− − −	− − −	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− − −	− − −	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{0_1}	∘ 1 ∘	− 0 −	− 0 −	− − −	− − −	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{0_2}	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− − 0	− − −	− −	− −	1 0	c
s_{1_0}	− − −	1 ∘ ∘	− − −	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{1_1}	− 0 −	∘ 1 ∘	− − −	− 0 −	− 0 −	− − −	− − −	− −	1 0	− −	$b \vee$
s_{1_2}	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− − 0	− −	− −	0 1	$\neg c$
s_{2_0}	− − −	− − −	1 ∘ ∘	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{2_1}	− 0 −	− − −	∘ 1 ∘	− 0 −	− 0 −	− − −	− − −	− −	1 0	− −	$b \vee$
s_{2_2}	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− −	− −	1 0	c
s_{3_0}	0 − −	0 − −	0 − −	1 ∘ ∘	− − −	− − −	− − −	1 0	− −	− −	$a \vee$
s_{3_1}	− − −	− 0 −	− 0 −	∘ 1 ∘	− − −	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{3_2}	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− −	− −	0 1	$\neg c$
s_{4_0}	0 − −	0 − −	0 − −	− − −	1 ∘ ∘	− − −	− − −	1 0	− −	− −	$a \vee$
s_{4_1}	− − −	− 0 −	− 0 −	− − −	∘ 1 ∘	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{4_2}	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− −	− −	1 0	c
s_{5_0}	0 − −	0 − −	0 − −	− − −	− − −	1 ∘ ∘	− − −	1 0	− −	− −	$a \vee$
s_{5_1}	− 0 −	− − −	− − −	− 0 −	− 0 −	∘ 1 ∘	− − −	− −	1 0	− −	$b \vee$
s_{5_2}	− − 0	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− −	− −	0 1	$\neg c$
s_{6_0}	0 − −	0 − −	0 − −	− − −	− − −	− − −	1 ∘ ∘	1 0	− −	− −	$a \vee$
s_{6_1}	− 0 −	− − −	− − −	− 0 −	− 0 −	− − −	∘ 1 ∘	− −	1 0	− −	$b \vee$
s_{6_2}	− − −	− − 0	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− −	− −	1 0	c
s_{7_0}	0 − −	0 − −	0 − −	− − −	− − −	− − −	− − −	1 ∘	− −	− −	a
s_{7_1}	− − −	− − −	− − −	0 − −	0 − −	0 − −	0 − −	∘ 1	− −	− −	$\neg a$
s_{8_0}	− 0 −	− − −	− − −	− 0 −	− 0 −	− − −	− − −	− −	1 ∘	− −	b
s_{8_1}	− − −	− 0 −	− 0 −	− − −	− − −	− 0 −	− 0 −	− −	∘ 1	− −	$\neg b$
s_{9_0}	− − −	− − 0	− − −	− − 0	− − −	− − 0	− − −	− −	− −	1 ∘	c
s_{9_1}	− − 0	− − −	− − 0	− − −	− − 0	− − −	− − 0	− −	− −	∘ 1	$\neg c$

.

The components of a satoku matrix are defined as follows:

- A selection row is an entire satoku matrix row, which represents a selection from a clause of the mapped SAT problem including the dependencies on all other selections.

  The selection row indexing s_{i_j} refers to conjunction j from clause i of the mapped SAT problem.

Selection row s_{0_0} of the example SM:

$$s_{0_0} \mid 1\circ\circ \mid --- \mid --- \mid 0-- \mid 0-- \mid 0-- \mid 0-- \parallel 0\,1 \mid -- \mid -- \mid$$

- satoku matrix subdivisions limited by horizontal lines represent clause sub-matrices S_i, containing one or more selection rows s_{i_j}.

  Clause sub-matrices are further divided by vertical lines into clause conflict sub-matrices $S_{i,f}$, where f has the same range of values as i.

  A clause conflict sub-matrix $S_{i,f}$ represents the dependencies between all selections from two SAT problem clauses S_i and S_f.

  The special clause conflict sub-matrix $S_{i,i}$ is called clause identity sub-matrix.

  Clause sub-matrix S_0 of the example SM:

s_{0_0}	$1\circ\circ$	$---$	$---$	$0--$	$0--$	$0--$	$0--$	$0\,1$	$--$	$--$
s_{0_1}	$\circ 1\circ$	$-0-$	$-0-$	$---$	$---$	$-0-$	$-0-$	$--$	$0\,1$	$--$
s_{0_2}	$\circ\circ 1$	$--0$	$---$	$--0$	$---$	$--0$	$---$	$--$	$--$	$1\,0$

  Clause conflict sub-matrix $S_{0,3}$ of the example SM highlighted in green:

s_{0_0}	$1\circ\circ$	$---$	$---$	**$0--$**	$0--$	$0--$	$0--$	$0\,1$	$--$	$--$
s_{0_1}	$\circ 1\circ$	$-0-$	$-0-$	**$---$**	$---$	$-0-$	$-0-$	$--$	$0\,1$	$--$
s_{0_2}	$\circ\circ 1$	$--0$	$---$	**$--0$**	$---$	$--0$	$---$	$--$	$--$	$1\,0$

  Clause identity sub-matrix $S_{0,0}$ of the example SM highlighted in green:

s_{0_0}	**$1\circ\circ$**	$---$	$---$	$0--$	$0--$	$0--$	$0--$	$0\,1$	$--$	$--$
s_{0_1}	**$\circ 1\circ$**	$-0-$	$-0-$	$---$	$---$	$-0-$	$-0-$	$--$	$0\,1$	$--$
s_{0_2}	**$\circ\circ 1$**	$--0$	$---$	$--0$	$---$	$--0$	$---$	$--$	$--$	$1\,0$

- A clause segment $cs_{i_j,f}$ denotes the intersection of selection row s_{i_j} and clause conflict sub-matrix $S_{i,f}$.

  Clause segment $cs_{0_1,2}$ of the example SM highlighted in green:

$$s_{0_1} \mid \circ 1\circ \mid -0- \mid \textbf{-0-} \mid --- \mid --- \mid -0- \mid -0- \parallel -- \mid 0\,1 \mid -- \mid$$

- A dash (-) is merely a substitution for the boolean value 1, if there is more than one matrix cell containing the value 1 present in a clause segment of a selection row. It designates a possible selection of the corresponding selection row.
- The row labeled P is not part of the matrix, it shows the status of globally possible (- or +), required (1) and impossible (0) selections.
- The double lines are just visual aids to separate semantically different groups (e.g., core problem clauses and variables). They have the same meaning as a single line.
- The right margin is used for comments which can be completely ignored.

5. Adjacency Matrix Properties

Given a set of clauses C and a set of variables V mapped to a set W of 2-literal clauses $w_j = (v_j \vee \neg v_j), v_j \in V, j = (0, 1, \ldots, |V| - 1)$, the row count L is:

$$\begin{aligned} L &= \sum_{i=0}^{m-1} |c_i| + \sum_{j=0}^{n-1} |w_j| \\ & \quad m = |C|, c \in C, n = |W|, w \in W \end{aligned}$$

A matrix cell refers to the value in the matrix at a specific position. For the adjacency matrix cell properties in this section, the standard index scheme for row t and column u is used: $t, u = (0, 1, \ldots, L - 1)$. Rows are denoted by r_t, matrix cells are denoted by $c_{t,u}$.

Each matrix row consists of L matrix cells. The SM consists of L selection rows.

Therefore the space requirements of the mapping algorithm are $\mathcal{O}(L^2)$.

- An inverted adjacency matrix is necessarily symmetrical (mirrored at the matrix diagonal). This property follows from mapping an undirected graph to an inverted adjacency matrix. This property also holds for the satoku matrix.
- The row and column indexes of a matrix cell are equivalent to the row indexes of the related rows. E.g., cell $c_{3,5}$ refers to row r_3 and row r_5.
- If the value of the matrix cell $c_{t,u}$ is zero (0), it is said that the selection of row r_t depends on (conflicts with) the selection of row r_u, or that selection of row r_t makes selection of row r_u impossible.

  Due to symmetry, it can also be said, that selection of row r_u depends on (conflicts with) the selection of row r_t.

- If the value of the matrix cell $c_{t,u}$ is one (1), it is said that the selection of row r_t is independent from the selection of row r_u. In the context of row r_t, the selection of row r_u is possible.

  Due to symmetry, it can also be said, that the selection of row r_u is independent from the selection of row r_t.

6. Classification of Boolean Values

In the context of selection rows, a matrix cell is denoted as c_{i_j,f_g}, corresponding to selection rows s_{i_j} and s_{f_g}.

The boolean value 0 in a matrix cell c_{i_j,f_g} signifies that selection s_{f_g} is impossible in the context of selection row s_{i_j}.

The boolean value instances of 1 in the SM are classified based on the number of occurrences in a clause segment of a selection row.

- If a clause segment in a selection row contains more than one matrix cell with the boolean value 1, the values are replaced by a dash (-) for display and called *soft one*. Note that the truth value of a *soft one* is still `T`, the substitution is merely ornamental.

- If a clause segment $cs_{i_j,f}$ in a selection row s_{i_j} contains exactly one matrix cell c_{i_j,f_g} with the boolean value 1, the value is unchanged and called a *hard one*. The corresponding selection row s_{f_g} becomes a required (or necessary) selection for selection row s_{i_j}.

Example of unclassified selection row s_{0_0}:

$$s_{0_0} \,|\, 1\,0\,0 \,|\, 1\,1\,1 \,|\, 1\,1\,1 \,|\, 0\,1\,1 \,|\, 0\,1\,1 \,|\, 0\,1\,1 \,|\, 0\,1\,1 \,\|\, 0\,1 \,|\, 1\,1 \,|\, 1\,1 \,|$$

transformed to classified selection row s_{0_0}:

$$s_{0_0} \,|\, 1 \circ \circ \,|\, - - - \,|\, - - - \,|\, 0 - - \,|\, 0 - - \,|\, 0 - - \,|\, 0 - - \,\|\, 0\,1 \,|\, - - \,|\, - - \,|$$

7. Requirements Update Algorithm

Following the consequences of potential selections can be interpreted as simple conflict propagation. However, a more flexible interpretation is that of incrementally refining partial assignments, effectively resulting in a partial distributive expansion of the problem clauses.

The satoku matrix is used as a dynamic data structure. It is therefore necessary to define the operations in a manner as to preserve the properties of an inverted adjacency matrix.

In order to facilitate detection of *soft one* $\rightarrow$ *hard one* transitions, a *soft one* is represented by the value 2.

Assignment of a value z to a variable x is denoted as $x \leftarrow z$.

7.1 Selection Logic

The logic of selections involves the truth values `T`, `F` and the logical functions AND and (implicitely) single selection (X1).

Table 1 shows how matrix cell values map to truth values.

Matrix Cell Value	Truth Value
0	`F`
1	`T`
2	`T`

Table 1. Mapping of Matrix Cell Values to Truth Values

The *hard*/*soft* property for truth value `T` is determined contextually and has different semantics in regard to the required updates and the relationship between selection rows. However, for the purpose of logical operations both *hard one*s and *soft one*s represent the truth value `T`.

7.2 Mirror Property

Each matrix cell $c_{t,u}$ has a mirror cell $c_{u,t}$, which must be kept synchronized. Cells on the matrix diagonal are their own mirror, which makes synchronization trivial since they are obviously always synchronized to themselves.

RT: $\mathcal{O}(1)$

7.3 Unselectable Rows, Unsatisfiability

- If it so happens, that any clause segment $cs_{i_j,f}$ of a selection row s_{i_j} is filled entirely with zeroes, s_{i_j} can no longer be selected at all, i.e., it becomes unselectable. Therefore the entire row s_{i_j} is filled with zeroes. Due to the mirror cell requirements, column sc_{i_j} is also filled with zeroes. The selection of row s_{i_j} is called a contradiction.

  RT: $\mathcal{O}(n)$

- If all selection rows s_{i_j} in a clause sub-matrix S_i become unselectable, the corresponding propositional problem is unsatisfiable.

  RT: $\mathcal{O}(1)$

7.4 *Soft*/*Hard* Property, Superset Rows

- If there is a selection row s_{i_j}, with a *hard one* in cell c_{i_j,f_g}, it follows, that if the selection represented by s_{i_j} is made then the selection represented by s_{f_g} must also be made. This is necessary to preserve satisfiability.

  It is said, that selection row s_{i_j} requires selection row s_{f_g} (necessary selection). Therefore, selection row s_{i_j} is called a superset row of s_{f_g}.

- If it so happens, that a clause segment $cs_{i_j,f}$ of a selection row s_{i_j} is filled with zeroes except for a single *soft one* at c_{i_j,f_g}, the *soft one* becomes a *hard one* and the conflicts of the required selection row s_{f_g} are added to row s_{i_j}.

  All values in the matrix cells c_{i_j,x_y} of selection row s_{i_j} are replaced by the logical AND of the value in c_{i_j,x_y} and the value in c_{f_g,x_y} from selection row s_{f_g}:

$$\begin{aligned} & s_{i_j} \wedge s_{f_g} \\ = \quad & \forall x, y : c_{i_j,x_y} \leftarrow c_{i_j,x_y} \wedge c_{f_g,x_y}, \\ & i, f, x \in \{0, 1, \ldots, |P| - 1\}, \\ & j \in \{0, 1, \ldots, |C_i| - 1\}, \\ & g \in \{0, 1, \ldots, |C_f| - 1\}, \\ & y \in \{0, 1, \ldots, |C_x| - 1\} \end{aligned}$$

  This effectively adds all conflicts (zeroes) from s_{f_g} to s_{i_j}.

  RT: $\mathcal{O}(n)$

- If new conflicts are added to a selection row (matrix cell value changes from 1 or 2 to 0), all superset rows must be updated accordingly.

7.5 Serialized Requirement Update Algorithm

To facilitate analysis of run-time behavior, a requirements update algorithm is constructed, which serializes the update steps.

For a k-SAT problem P with m clauses C of fixed size k, the number of selection rows r in the satoku matrix SM is:

$$r = k \cdot m.$$

The number of matrix cells is r^2.

7.5.1 Global Status

Clause information (index, offset, size) is stored in array `clauses`.

Matrix row information is stored in array `rows` consisting of an array of bytes (matrix cells) for each row. A matrix cell can have the values $0, 1, 2$ signifying a conflict(0), a *hard one* (1) or a *soft one* (2).

The global SM status is tracked with flag arrays and request lists.

The array `row_status` contains a flag for each matrix row s_{i_j}. The initial value is 2 (*soft one*) for a posssible selection. If a selection row s_{i_j} becomes unselectable, the corresponding flag $\texttt{row\_status}_{i_j}$ becomes 0.

The list `zero_updates` holds the required mirror cell zero transitions. Duplicate zero update requests are implicitely avoided in function `zero_change`.

The array `row_update_flags` contains a flag for each matrix row. The initial value is 0. If the requirements for the row must be updated, the flag becomes 1. The flags are used to prevent unnecessary duplicate updates.

The list `row_updates` holds the requirement update info for each row in the order of detection.

The array `superset_update_flags` contains a flag for each matrix row. The initial value is 0. If the superset requirements for a row must be updated, the flag becomes 1. The flags are used to prevent unnecessary duplicate updates.

The list `superset_updates` holds the superset requirement update info for each row in the order of detection.

Space requirements for the data structures are shown in table 2.

Data Structure	**Space**
`clauses`[m]	SPACE(m)
`rows`[r]	SPACE(r^2)
`row_status`[r]	SPACE(r)
`zero_updates`[] (list of requests)	SPACE(r^2)
`row_update_flags`[r] (flags)	SPACE(r)
`row_updates`[] (list of requests)	SPACE(r)
`superset_update_flags`[r] (flags)	SPACE(r)
`superset_updates`[] (list of requests)	SPACE(r)

Table 2. Space Requirements for Auxiliary Data Structures

7.5.2 Get Selection Request

Get the offset of a selection request. I.e., the index of a single *soft one* in a clause segment $cs_{i_j,f}$ of a selection row.

```
return value: index of selection request or -1
function get_selection_request(row, f):                    O(k)
  sel_req_index ← -1
  size ← clauses[f].size
  ci ← clauses[f].offset
  for g in range(size):                                    O(k)
    if row[ci + g] > 1:
      if sel_req_index ≥ 0:
        sel_req_index ← -1
        break
      sel_req_index ← g
  return sel_req_index
```

7.5.3 One Counts

Get *hard one* and *soft one* counts for a clause segment.

```
return values: soft one count, hard one count
function one_counts(row, f):                               O(k)
  soft_count ← 0
  hard_count ← 0
  size ← clauses[f].size
  ci ← clauses[f].offset
  for g in range(size):                                    O(k)
    value ← row[ci + g]
    if value > 1:
      soft_count ← soft_count + 1
    elif value > 0:
      hard_count ← hard_count + 1
  return soft_count, hard_count
```

7.5.4 Row Update Request

A `row_update_request` is issued, when the requirements of a selection row must be updated.

```
procedure row_update_request(i, j):                                    O(1)
  ri ← clauses[i].offset + j
  if row_status[ri] = 0:
    return
  if row_update_flags[ri] = 1:
    return
  mark row in row_update_flags:
    row_update_flags[ri] ← 1
  add update request to row_updates
```

7.5.5 Superset Update Request

A `superset_update_request` is issued, when the requirements for the superset rows of a selection row must be updated.

```
procedure superset_update_request(i, j):                               O(1)
  ri ← clauses[i].offset + j
  if row_status[ri] = 0:
    return
  if row_update_flags[ri] = 1:
    return
  if superset_update_flags[ri] = 1:
    return
  mark row in superset_update_flags:
    superset_update_flags[ri] ← 1
  add superset update request to superset_updates
```

7.5.6 Kill Row

When all matrix cells of a clause segment of a selection row become 0 (contradiction), the row becomes unselectable and is filled with zeroes.

Since unselectable rows decrease the problem size, early detection of contradictions is desirable.

```
procedure kill_row(i, j):                                              O(r)
  ri ← clauses[i].offset + j
  mark row as inactive:
    row_status[ri] ← 0
  if all rows in clause are disabled:                                  O(k)
    raise UNSAT
  for clause in clauses:                                               O(m)
    f ← clause.index
    for g in range(clause.size):                                       O(k)
      zero_change(f, g, i, j, immediate=True)                          O(k)
```

7.5.7 Zero Change

Apply zero transition to a matrix cell.

`row_update_request` and `superset_update_request` are issued as appropriate.

Early contradiction detection is implemented, since it decreases problem size. However, the running time for killing a row is not included in the overall running time estimate.

```
return values: killed flag, changed flag
function zero_change(i, j, f, g, immediate=False):                    O(k)
  ri ← clauses[i].offset + j
  ci ← clauses[f].offset + g
  killed ← False
  changed ← False
  if row_status[ri] = 0:
    return killed, changed
  row ← rows[ri]
  if row[ci] ≠ 0:
    changed ← True
    row[ci] ← 0
    soft_count, hard_count ← one_counts(row, f)                       O(k)
    killed ← (soft_count + hard_count) = 0
    if killed:
      kill_row(i, j)                                                  O(r)
      return killed, changed
    if soft_count = 1:
      row_update_request(i, j)                                        O(1)
    else:
      superset_update_request(i, j)                                   O(1)
    if rows[ci][ri] ≠ 0:
      if immediate:
        return zero_change(f, g, i, j)                                O(k)
      else:
        if row_status[ci] ≠ 0:
          soft_count, hard_count ← one_counts(rows[ci], i)            O(k)
          mkilled ← (soft_count + hard_count) = 1
          if mkilled:
            kill_row(f, g)                                            O(r)
          else:
            add zero request for mirror cell (zero_updates)
  return killed, changed
```

7.5.8 Update Row

Update required selections of a selection row.

The matrix cell values 1(*hard one*) and 2(*soft one*) are equivalent to the truth value `T`. The matrix cell value 0 is equivalent to the truth value `F` (see table 1).

If a clause segment $cs_{i_j,f}$ of a selection row s_{i_j} (row) contains a single truth value `T` in matrix cell s_{i_j,f_g}, the corresponding selection row s_{f_g} (rq_row) is required. In the context of this algorithm, the required selection must be updated, if the single truth value is a *soft one* (2).

Table 3 shows how the matrix cell values of the required selection row s_{f_g} (rq_row) are incorporated into the selection row s_{i_j} (row). *ci* denotes the matrix cell index.

row[ci]	rq_row[ci]	row[ci]'
row[ci]	$\leq$ rq_row[ci]	row[ci]
2	1	1
1, 2	0	`zero_change`(i, j, ...)

Table 3. Logical AND of selection rows

The result is equivalent to a logical AND of selection rows s_{i_j}, s_{f_g}:

$$s'_{i_j} = s_{i_j} \wedge s_{f_g}.$$

Note, that a selection row with a contradiction is not considered changed, since there can no longer be any superset rows after the row and column are filled with zeroes.

```
return value: row_changed
function update_row(i, j):                                              O(r^3)
  row_changed ← False
  ri ← clauses[i].offset + j
  if row_status[ri] = 0:
    row_update_flags[ri] ← 0
    return row_changed
  if row_update_flags[ri] = 0:
    return row_changed

  row ← rows[ri]
  row_size ← len(row)
  updated ← True
  while updated:                                                        O(m^2 · r)
    updated ← False
    for clause in clauses:                                              O(m)
      f ← clause.index
      if f = i:
        // skip identity clause segment
        continue
      g ← get_selection_request(row, f)                                 O(k)
      if g ≥ 0:
        rq_row ← rows[clause.offset + g]
        for ci in range(row_size):                                      O(r)
          if rq_row[ci] < row[ci]:
            if rq_row[ci] = 0:
              rq_f = row_clause_map[ci]
              rq_g = row_literal_map[ci]
              killed, changed = zero_change(i, j, rq_f, rq_g)           O(k)
              if killed:
                row_changed = False
                row_update_flags[ri] ← 0
                return row_changed
              row_changed ← row_changed ∨ changed
              updated ← True
            else:
              // this will change the soft one
              // for row[ci] to a hard one
              row[ci] ← rq_row[ci]
  row_update_flags[ri] ← 0
  return row_changed
```

7.5.9 Update Zeroes

Handle pending zero changes.

The absolute maximum of pending zero changes is $\frac{r^2}{2}$. A pending zero change cannot trigger another zero change. It can, however, trigger a `row_update_request`, `superset_update_request` or `kill_row`.

```
procedure update_zeroes()                                   O(r^2)
  if len(zero_updates) = 0:
    return
  zuc ← zero_updates
  zero_updates ← []
  for zu in zuc:                                            O(r^2)
    zero_change(zu.i, zu.j, zu.f, zu.g)                     O(k)
```

7.5.10 Update Superset Rows

Change *hard one* to *soft one* in superset rows and issue update requests.

```
procedure update_superset_rows(i, j):                       O(r)
  ci ← clauses[i].offset + j
  if row_status[ci] = 0:
    superset_update_flags[ci] ← 0
    return
  if superset_update_flags[ci] = 0:
    return
  for ri, row in enumerate(rows):                           O(r)
    if ri = ci:
      continue
    if row[ci] = 1:
      row[ci] ← 2
      f, g ← clause_map[ri]
      row_update_request(f, g)
  superset_update_flags[ci] ← 0
```

7.5.11 Update Supersets

Handle pending superset row update requests.

```
procedure update_supersets()                                O(r^2)
  suc ← superset_updates
  superset_updates ← []
  for req in suc:                                           O(r^2)
    update_superset_rows(req.i, req.j)                      O(r)
```

7.5.12 Update Rows

Propagate conflicts throughout the SM.

```
procedure update_rows()                                          | O(r^5)
  update_zeroes()                                                | O(r^2)
  while row_updates:                                             | O(r^5)
    ruc ← row_updates
    row_updates ← []
    for req in ruc:                                              | O(r^4)
      row_changed ← update_row(req.i, req.j)                     | O(r^3)
      if row_changed:
        superset_update_request(req.i, req.j)
    update_zeroes()                                              | O(r^2)
    update_supersets()                                           | O(r^2)
```

7.6 Worst Case Running Time

The functions `zero_change` and `kill_row` form a recursive loop. The maximum recursion depth is r, when all selection rows are killed. The worst case running time is therefore $\mathcal{O}(r^2)$.

The running time estimates given for the row update procedures are just theoretical upper bounds to show that the algorithm runs in polynomial time. The actual worst case for the reference implementation occurs, when a 2-SAT problem in SNF notation is mapped and all clauses become identical.

Measurements with the reference implementation show an overall run time behavior of $\mathcal{O}(r^2)$. I.e., the requirements update algorithm is actually linear over the satoku matrix.

The reason is that when all clauses become identical, only the k rows of the first clause actually update with at most $m-1$ other rows each, $k \cdot (m-1)$. All rows in other clauses only update once with one of the rows from the first clause, $k \cdot (m-1)$, giving a total of $k \cdot (m-1) + k \cdot (m-1) = 2 \cdot k \cdot (m-1)$ updates. See Row Updates for a step-by-step example.

There is also plenty of room to optimize the requirements update algorithm by using the same row data for identical rows (rows that require each other).

8. Conflicts

See Indirect Conflicts for a discussion of conflicts.

todo

9. Solutions

See 2-SAT Solutions for distributive expansion and the set of minimal solutions for a boolean problem. It also describes alternative methods of retrieving a set of (partial) solutions.

todo

Appendix A. Clause Vectors

Clause vectors are rows of a clause variable matrix. Handbook of Satisfiability Chatper 11.

Appendix B. Rationale

This section describes some of the design choices more detailed.

B.1. Allow Negated and Unnegated Variables in the Same Clause

Redundancies appear in many shapes and sizes. The requirement of removing the most trivial ones accomplishes nothing. The simple problem:

$$(p \vee q) \wedge (\neg p \vee \neg q)$$

is reduced in the satoku matrix to an equivalence of all clauses:

P	$--$	$--$	$--$	$--$	
s_{0_0}	1 ∘	0 1	1 0	0 1	$p \vee$
s_{0_1}	∘ 1	1 0	0 1	1 0	q
s_{1_0}	0 1	1 ∘	0 1	1 0	$\neg p \vee$
s_{1_1}	1 0	∘ 1	1 0	0 1	$\neg q$
s_{2_0}	1 0	0 1	1 ∘	0 1	p
s_{2_1}	0 1	1 0	∘ 1	1 0	$\neg p$
s_{3_0}	0 1	1 0	0 1	1 ∘	q
s_{3_1}	1 0	0 1	1 0	∘ 1	$\neg q$

.

After removing the redundancies, the satoku matrix presents as:

P	$--$	
s_{0_0}	1 ∘	p
s_{0_1}	∘ 1	$\neg p$

,

which shows, that the original problem was just an elaborate way of specifying $(p \vee \neg p)$:

$$(p \equiv \neg q) \wedge (\neg p \equiv q)$$
$$\Rightarrow (p \vee q) \wedge (\neg p \vee \neg q) \equiv (p \vee \neg p) \wedge (\neg p \vee p)$$
$$\Rightarrow (p \vee \neg p)$$

.

This technique could be used to replace all variable clauses $(p \vee \neg p)$ with 2 clauses $(p \vee q) \wedge (\neg p \vee \neg q)$, where q is a new variable, that does not appear otherwise in the formula.

While this would then satisfy the formal requirements for a CNF formula, it does seem a bit silly. Expanding the 2-literal clauses further to 3-literal clauses would make the Kafkaesque bureaucracy perfect:

$$\begin{array}{l}
(\ \ p \vee \ \ q \vee \neg r) \wedge \\
(\ \ p \vee \ \ q \vee \ \ r) \wedge \\
(\neg p \vee \neg q \vee \neg s) \wedge \\
(\neg p \vee \neg q \vee \ \ s)
\end{array}$$

Mapped to a plain satoku matrix, the monstrosity appears as a real problem:

P	− − −	− − −	− − −	− − −	− −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	− −	
s_{0_1}	∘ 1 ∘	− − −	− 0 −	− 0 −	− −	1 0	− −	− −	
s_{0_2}	∘ ∘ 1	− − 0	− − −	− − −	− −	− −	0 1	− −	
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	− −	
s_{1_1}	− − −	∘ 1 ∘	− 0 −	− 0 −	− −	1 0	− −	− −	
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − −	− −	− −	1 0	− −	
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	− −	
s_{2_1}	− 0 −	− 0 −	∘ 1 ∘	− − −	− −	0 1	− −	− −	
s_{2_2}	− − −	− − −	∘ ∘ 1	− − 0	− −	− −	− −	0 1	
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− −	
s_{3_1}	− 0 −	− 0 −	− − −	∘ 1 ∘	− −	0 1	− −	− −	
s_{3_2}	− − −	− − −	− − 0	∘ ∘ 1	− −	− −	− −	1 0	
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	− −	p
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	− −	$\neg p$
s_{5_0}	− − −	− − −	− 0 −	− 0 −	− −	1 ∘	− −	− −	q
s_{5_1}	− 0 −	− 0 −	− − −	− − −	− −	∘ 1	− −	− −	$\neg q$
s_{6_0}	− − 0	− − −	− − −	− − −	− −	− −	1 ∘	− −	r
s_{6_1}	− − −	− − 0	− − −	− − −	− −	− −	∘ 1	− −	$\neg r$
s_{7_0}	− − −	− − −	− − 0	− − −	− −	− −	− −	1 ∘	s
s_{7_1}	− − −	− − −	− − −	− − 0	− −	− −	− −	∘ 1	$\neg s$

,

Mapped to a satoku matrix with conflict bias:

P	$--0$	$--0$	$--0$	$--0$	$--$	$--$	$--$	$--$	
s_{0_0}	$1\circ\circ$	$1\,0\,0$	$0\,1\,0$	$0\,1\,0$	$1\,0$	$0\,1$	$--$	$--$	
s_{0_1}	$\circ 1\circ$	$0\,1\,0$	$1\,0\,0$	$1\,0\,0$	$0\,1$	$1\,0$	$--$	$--$	
s_{0_2}	$\circ\circ\circ$	$0\,0\,0$	$0\,0\,0$	$0\,0\,0$	$0\,0$	$0\,0$	$0\,0$	$0\,0$	
s_{1_0}	$1\,0\,0$	$1\circ\circ$	$0\,1\,0$	$0\,1\,0$	$1\,0$	$0\,1$	$--$	$--$	
s_{1_1}	$0\,1\,0$	$\circ 1\circ$	$1\,0\,0$	$1\,0\,0$	$0\,1$	$1\,0$	$--$	$--$	
s_{1_2}	$0\,0\,0$	$\circ\circ\circ$	$0\,0\,0$	$0\,0\,0$	$0\,0$	$0\,0$	$0\,0$	$0\,0$	
s_{2_0}	$0\,1\,0$	$0\,1\,0$	$1\circ\circ$	$1\,0\,0$	$0\,1$	$1\,0$	$--$	$--$	
s_{2_1}	$1\,0\,0$	$1\,0\,0$	$\circ 1\circ$	$0\,1\,0$	$1\,0$	$0\,1$	$--$	$--$	
s_{2_2}	$0\,0\,0$	$0\,0\,0$	$\circ\circ\circ$	$0\,0\,0$	$0\,0$	$0\,0$	$0\,0$	$0\,0$	
s_{3_0}	$0\,1\,0$	$0\,1\,0$	$1\,0\,0$	$1\circ\circ$	$0\,1$	$1\,0$	$--$	$--$	
s_{3_1}	$1\,0\,0$	$1\,0\,0$	$0\,1\,0$	$\circ 1\circ$	$1\,0$	$0\,1$	$--$	$--$	
s_{3_2}	$0\,0\,0$	$0\,0\,0$	$0\,0\,0$	$\circ\circ\circ$	$0\,0$	$0\,0$	$0\,0$	$0\,0$	
s_{4_0}	$1\,0\,0$	$1\,0\,0$	$0\,1\,0$	$0\,1\,0$	$1\circ$	$0\,1$	$--$	$--$	p
s_{4_1}	$0\,1\,0$	$0\,1\,0$	$1\,0\,0$	$1\,0\,0$	$\circ 1$	$1\,0$	$--$	$--$	$\neg p$
s_{5_0}	$0\,1\,0$	$0\,1\,0$	$1\,0\,0$	$1\,0\,0$	$0\,1$	$1\circ$	$--$	$--$	q
s_{5_1}	$1\,0\,0$	$1\,0\,0$	$0\,1\,0$	$0\,1\,0$	$1\,0$	$\circ 1$	$--$	$--$	$\neg q$
s_{6_0}	$--0$	$--0$	$--0$	$--0$	$--$	$--$	$1\circ$	$--$	r
s_{6_1}	$--0$	$--0$	$--0$	$--0$	$--$	$--$	$\circ 1$	$--$	$\neg r$
s_{7_0}	$--0$	$--0$	$--0$	$--0$	$--$	$--$	$--$	$1\circ$	s
s_{7_1}	$--0$	$--0$	$--0$	$--0$	$--$	$--$	$--$	$\circ 1$	$\neg s$

,

the problem goes away, but 2 of the 3 additional variables remain as artifacts, which introduces even more tautologies than originally planned:

P	$--$	$--$	$--$	
s_{0_0}	$1\circ$	$--$	$--$	p
s_{0_1}	$\circ 1$	$--$	$--$	$\neg p$
s_{1_0}	$--$	$1\circ$	$--$	r
s_{1_1}	$--$	$\circ 1$	$--$	$\neg r$
s_{2_0}	$--$	$--$	$1\circ$	s
s_{2_1}	$--$	$--$	$\circ 1$	$\neg s$

.

Note, that MiniSat v1.14 actually makes 4 decisions for this problem, MiniSat v2.2.0 reports 1 decision and lingeling ats 57807c8f410a9e676816984a0ad0c410e485bcae reports no decisions.

B.2. CNF vs. ConDNF, Conflict Bias

Given the propositional formulas:

$$\begin{aligned} A = &\ \neg p \wedge q, \\ B = &\ \neg p \wedge \neg q \wedge r, \\ C = &\ p \vee A \vee B, \\ D = &\ p \vee q \vee r, \end{aligned}$$

a fully expanded truth table shows, that the propositional formulas C and D are logically equivalent:

p	q	r	$\neg p$	$\neg q$	$\neg r$	A	B	C	D
0	0	0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	1	1	1
0	1	0	1	0	1	1	0	1	1
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	1	1
1	0	1	0	1	0	0	0	1	1
1	1	0	0	0	1	0	0	1	1
1	1	1	0	0	0	0	0	1	1

Therefore, the following equivalence holds:

$$\begin{aligned} &(p \ \vee \ q \ \vee \ r) \\ \equiv &(p \ \vee \ (\neg p \wedge q) \ \vee \ (\neg p \wedge \neg q \wedge r)) \end{aligned}$$

Propositional formulas C and D are said to be logically equivalent.

Two clauses c_0, c_1 containing only literals, are said to be structurally equivalent $c_0 \overset{s}{=} c_1$, if both contain the same set of literals.

Two propositional formulas P_0, P_1 are said to be structurally equivalent $P_0 \overset{s}{=} P_1$, if both contain the same set of clauses. I.e. For each clause $c_i \in P_0$ there is a structurally equivalent clause $c_j \in P_1$ and for each clause $c_j \in P_1$ there is a structurally equivalent clause $c_i \in P_0$:

$$\begin{aligned} \forall c_i \exists c_j : c_i \overset{s}{=} c_j, c_i \in P_0, c_j \in P_1 \\ \forall c_j \exists c_i : c_i \overset{s}{=} c_j, c_i \in P_0, c_j \in P_1 \end{aligned}$$

A minimal DNF b_m is a disjunction of single literal conjunctions. E.g., $((p)\vee(q)\vee(r))$:

```
[ // OR
[ 1 _ _ ] // AND
[ _ 1 _ ] // AND
[ _ _ 1 ] // AND
]
  p q r
```

The set of variables V for b_m consists of all variables implied by the single literal conjunctions of a minimal DNF b_m.

The set of conjunctions over the power set of literals over V without the conjunctions evaluating to `F` is denoted as C_p. The conjunction c_p denotes a member of $C_p \cup \{\texttt{T}\}$. (It's really not that hard: c_p is either empty (`T`) or any conjunction of literals.)

The conflict bias set B is defined as the set of DNFs which are all logically equivalent to the minimal DNF b_m. A member of B is said to represent the minimal DNF b_m.

The expansion algorithm E is defined as:

1. Expand a conjunction $c_i \in b, b \in B$, that does not contain a variable $p \in V$, by replacing c_i with two conjunctions $c_j = c_i \wedge p$ and $c_k = c_i \wedge \neg p$

2. Remove all redundant conjunctions $c_j, c_i \neq c_j, c_i \overset{s}{=} c_j$.

Applying algorithm E recursively to a DNF $b \in B$ until the result of $E(b)$ is structurally equivalent to its input, $E(b) \overset{s}{=} b$, generates the fully expanded representative b_f of b_m. E.g.:

```
[ // OR
[ 1 0 0 ] // AND
[ 1 0 1 ] // AND
[ 1 1 0 ] // AND
[ 1 1 1 ] // AND
[ 0 1 0 ] // AND
[ 0 1 1 ] // AND
[ 0 0 1 ] // AND
]
  p q r
```

The fully expanded representative b_f of B can be reduced to a fully discriminated representative b_d by recursively applying the discrimination algorithm D, $b = D(b)$, until the result is structurally equivalent to the input $D(b) \stackrel{s}{=} b \Rightarrow b = b_d$:

1. If $c_i \in b$ and $c_j \in b$ are equivalent except for a single conflicting literal l, $c_i = c_p \wedge l, c_j = c_p \wedge \neg l$, the conflicting literal l can be removed from $c_i, c_i \mapsto c_i \backslash l$, since:
$$(p \wedge q) \vee (p \wedge \neg q) = (p) \vee (p \wedge \neg q)$$
2. Any conjunction c_j, which is equivalent to a conjunction $c_i \wedge c_p, c_i \neq c_j$ can be removed from $b, b \mapsto b \setminus c_j$, since:
$$\begin{array}{lcl} (p) \vee (p \wedge q) & = & (p) \vee (p) \\ (p) \vee (p) & = & (p) \end{array}$$

Examples for fully discriminated disjunctions of conjunctions:

```
[ // OR            |  [ // OR
[ 1 _ _ ] // AND  |  [ _ 1 _ ] // AND
[ 0 1 _ ] // AND  |  [ _ 0 1 ] // AND
[ 0 0 1 ] // AND  |  [ 1 0 0 ] // AND
]                 |  ]
  p q r           |    p q r
```

All members $b \in B$ can be reduced to the minimal DNF b_m by recursively applying the reduction algorithm R until no rule can be applied anymore and the result of $R(b)$ is structurally equivalent to its input, $R(b) \stackrel{s}{=} b$:

1. Apply discrimination algorithm D.
2. If c_i is a single literal conjunction (l), the literal $\neg l$ can be removed from a conjunction $c_j, c_j = \neg l \wedge c_p \Rightarrow c_j \mapsto c_j \setminus \neg l$, since:
$$(p) \vee (\neg p \wedge q) \quad = \quad (p) \vee (q)$$

B.2.1. Conflict Bias Basics

The fully discriminated representatives b_d can be interpreted as defining a conflict bias for selections from clauses. Replacing (r) in the DNF $((p) \vee (q) \vee (r))$ with $(\neg p \wedge \neg q \wedge r)$ puts more weight on the selection of r, since the conflicts for the selection are maximized.

Using a conflict bias instead of the minimal DNF has various semantically different but logically equivalent effects.

E.g., consider the following redundancy, that can be easily eliminated:

$$(p \vee q) \wedge (p \vee q \vee r),$$

mapped to clause vectors:

```
[ 1 1 _ ] // OR
[ 1 1 1 ] // OR
  p q r
```

Simply mapping it to a satoku matrix is pretty straightforward:

P	−−	−−−	−−	−−	−−	
s_{0_0}	1 ∘	−−−	1 0	−−	−−	$p \vee$
s_{0_1}	∘ 1	−−−	−−	1 0	−−	q
s_{1_0}	−−	1 ∘ ∘	1 0	−−	−−	$p \vee$
s_{1_1}	−−	∘ 1 ∘	−−	1 0	−−	$q \vee$
s_{1_2}	−−	∘ ∘ 1	−−	−−	1 0	r
s_{2_0}	−−	−−−	1 ∘	−−	−−	p
s_{2_1}	0 1	0 −−	∘ 1	1 0	−−	$\neg p$
s_{3_0}	−−	−−−	−−	1 ∘	−−	q
s_{3_1}	1 0	− 0 −	1 0	∘ 1	−−	$\neg q$
s_{4_0}	−−	−−−	−−	−−	1 ∘	r
s_{4_1}	−−	−− 0	−−	−−	∘ 1	$\neg r$

.

However, applying the following conflict bias:

```
[ // OR
[ 1 _ _ ] // AND
[ 0 1 _ ] // AND
]
[
[ 1 _ _ ] // AND
[ 0 1 _ ] // AND
[ 0 0 1 ] // AND
]
  p q r
```

narrows the possible choices maximally:

P	$--$	$--0$	$--$	$--$	$--$	
s_{0_0}	$1\circ$	$1\,0\,0$	$1\,0$	$--$	$--$	$p\vee$
s_{0_1}	$\circ\,1$	$0\,1\,0$	$0\,1$	$1\,0$	$--$	q
s_{1_0}	$1\,0$	$1\circ\circ$	$1\,0$	$--$	$--$	$p\vee$
s_{1_1}	$0\,1$	$\circ\,1\circ$	$0\,1$	$1\,0$	$--$	$q\vee$
s_{1_2}	$0\,0$	$\circ\circ\circ$	$0\,0$	$0\,0$	$0\,0$	r
s_{2_0}	$1\,0$	$1\,0\,0$	$1\circ$	$--$	$--$	p
s_{2_1}	$0\,1$	$0\,1\,0$	$\circ\,1$	$1\,0$	$--$	$\neg p$
s_{3_0}	$--$	$--0$	$--$	$1\circ$	$--$	q
s_{3_1}	$1\,0$	$1\,0\,0$	$1\,0$	$\circ\,1$	$--$	$\neg q$
s_{4_0}	$--$	$--0$	$--$	$--$	$1\circ$	r
s_{4_1}	$--$	$--0$	$--$	$--$	$\circ\,1$	$\neg r$

,

revealing r as completely redundant, leaving only a choice between p and q, :

P	$--$	$--$	$--$	$--$	
s_{0_0}	$1\circ$	$1\,0$	$--$	$--$	$p\vee$
s_{0_1}	$\circ\,1$	$0\,1$	$1\,0$	$--$	q
s_{1_0}	$1\,0$	$1\circ$	$--$	$--$	p
s_{1_1}	$0\,1$	$\circ\,1$	$1\,0$	$--$	$\neg p$
s_{2_0}	$--$	$--$	$1\circ$	$--$	q
s_{2_1}	$1\,0$	$1\,0$	$\circ\,1$	$--$	$\neg q$
s_{3_0}	$--$	$--$	$--$	$1\circ$	r
s_{3_1}	$--$	$--$	$--$	$\circ\,1$	$\neg r$

.

Note that the logical equivalence between the remaining clause ($p\vee q$) and the variable clause ($p\vee\neg p$) also means that no choice from the clauses is necessary.

B.2.2. Conflict Bias Variations

Complementary conflict bias priorities for clauses S_0, S_1:

P	−−	−−−	−−	−−	−−	
s_{0_0}	1 ∘	−−−	1 0	−−	−−	$p \vee$
s_{0_1}	∘ 1	−− 0	0 1	1 0	−−	q
s_{1_0}	−−	1 ∘ ∘	−−	−−	1 0	$r \vee$
s_{1_1}	−−	∘ 1 ∘	−−	1 0	0 1	$q \vee$
s_{1_2}	1 0	∘ ∘ 1	1 0	0 1	0 1	p
s_{2_0}	1 0	−−−	1 ∘	−−	−−	p
s_{2_1}	0 1	−− 0	∘ 1	1 0	−−	$\neg p$
s_{3_0}	−−	−− 0	−−	1 ∘	−−	q
s_{3_1}	1 0	− 0 −	1 0	∘ 1	−−	$\neg q$
s_{4_0}	−−	1 0 0	−−	−−	1 ∘	r
s_{4_1}	−−	0 −−	−−	−−	∘ 1	$\neg r$

.

Note that selecting s_{0_0} leaves the option to select q open. Whereas selecting s_{1_2} does not allow selecting q. this is a good example why a selection from a clause is not the same as making a decision for a variable (equivalent to making a selection from a variable clause).

Uniform conflict bias priorities for clauses S_0, S_1 by minimal occurence of a variable:

P	−−	−−−	−−	−−	−−	
s_{0_0}	1 ∘	−− 0	−−	1 0	−−	$q \vee$
s_{0_1}	∘ 1	− 0 −	1 0	0 1	−−	p
s_{1_0}	−−	1 ∘ ∘	−−	−−	1 0	$r \vee$
s_{1_1}	1 0	∘ 1 ∘	−−	1 0	0 1	$q \vee$
s_{1_2}	0 1	∘ ∘ 1	1 0	0 1	0 1	p
s_{2_0}	−−	−−−	1 ∘	−−	−−	p
s_{2_1}	1 0	−− 0	∘ 1	1 0	−−	$\neg p$
s_{3_0}	1 0	−− 0	−−	1 ∘	−−	q
s_{3_1}	0 1	− 0 −	1 0	∘ 1	−−	$\neg q$
s_{4_0}	−−	1 0 0	−−	−−	1 ∘	r
s_{4_1}	−−	0 −−	−−	−−	∘ 1	$\neg r$

.

Appendix C. Satoku Matrix Details

Mapping the following CNF 3-SAT formula:

$$
\begin{array}{l}
(\neg a \vee \neg b \vee \ \ c) \wedge \\
(\neg a \vee \ \ b \vee \neg c) \wedge \\
(\neg a \vee \ \ b \vee \ \ c) \wedge \\
(\ \ a \vee \neg b \vee \neg c) \wedge \\
(\ \ a \vee \neg b \vee \ \ c) \wedge \\
(\ \ a \vee \ \ b \vee \neg c) \wedge \\
(\ \ a \vee \ \ b \vee \ \ c)
\end{array}
$$

to an inverted adjacency matrix looks like this:

$$
\left[\begin{array}{cccccccccccccccccccc}
1&0&0&1&1&1&1&1&1&0&1&1&0&1&1&0&1&1&0&1&1\\
0&1&0&1&0&1&1&0&1&1&1&1&1&1&1&1&0&1&1&0&1\\
0&0&1&1&1&0&1&1&1&1&1&0&1&1&1&1&1&0&1&1&1\\
1&1&1&1&0&0&1&1&1&0&1&1&0&1&1&0&1&1&0&1&1\\
1&0&1&0&1&0&1&1&1&1&0&1&1&0&1&1&1&1&1&1&1\\
1&1&0&0&0&1&1&1&0&1&1&1&1&1&0&1&1&1&1&1&0\\
1&1&1&1&1&1&1&0&0&0&1&1&0&1&1&0&1&1&0&1&1\\
1&0&1&1&1&1&0&1&0&1&0&1&1&0&1&1&1&1&1&1&1\\
1&1&1&1&1&0&0&0&1&1&1&0&1&1&1&1&1&0&1&1&1\\
0&1&1&0&1&1&0&1&1&1&0&0&1&1&1&1&1&1&1&1&1\\
1&1&1&1&0&1&1&0&1&0&1&0&1&1&1&1&0&1&1&0&1\\
1&1&0&1&1&1&1&1&0&0&0&1&1&1&0&1&1&1&1&1&0\\
0&1&1&0&1&1&0&1&1&1&1&1&1&0&0&1&1&1&1&1&1\\
1&1&1&1&0&1&1&0&1&1&1&1&0&1&0&1&0&1&1&0&1\\
1&1&1&1&1&0&1&1&1&1&1&0&0&0&1&1&1&0&1&1&1\\
0&1&1&0&1&1&0&1&1&1&1&1&1&1&1&1&1&0&0&1&1\\
1&0&1&1&1&1&1&1&1&1&0&1&1&0&1&0&1&0&1&1&1\\
1&1&0&1&1&1&1&1&0&1&1&1&1&1&0&0&0&1&1&1&0\\
0&1&1&0&1&1&0&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
1&0&1&1&1&1&1&1&1&1&0&1&1&0&1&1&1&1&0&1&0\\
1&1&1&1&1&0&1&1&1&1&1&0&1&1&1&1&1&0&0&0&1
\end{array}\right]
$$

To further visualize the conflict situation, clause constraints are added and *soft one*s are replaced by dash (-). After this rule is applied, the satoku matrix looks like this:

s_{0_0}	$1\circ\circ$	$---$	$---$	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{0_1}	$\circ 1\circ$	$-0-$	$-0-$	$---$	$---$	$-0-$	$-0-$	$\neg b \vee$
s_{0_2}	$\circ\circ 1$	$--0$	$---$	$--0$	$---$	$--0$	$---$	c
s_{1_0}	$---$	$1\circ\circ$	$---$	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{1_1}	$-0-$	$\circ 1\circ$	$---$	$-0-$	$-0-$	$---$	$---$	$b \vee$
s_{1_2}	$--0$	$\circ\circ 1$	$--0$	$---$	$--0$	$---$	$--0$	$\neg c$
s_{2_0}	$---$	$---$	$1\circ\circ$	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{2_1}	$-0-$	$---$	$\circ 1\circ$	$-0-$	$-0-$	$---$	$---$	$b \vee$
s_{2_2}	$---$	$--0$	$\circ\circ 1$	$--0$	$---$	$--0$	$---$	c
s_{3_0}	$0--$	$0--$	$0--$	$1\circ\circ$	$---$	$---$	$---$	$a \vee$
s_{3_1}	$---$	$-0-$	$-0-$	$\circ 1\circ$	$---$	$-0-$	$-0-$	$\neg b \vee$
s_{3_2}	$--0$	$---$	$--0$	$\circ\circ 1$	$--0$	$---$	$--0$	$\neg c$
s_{4_0}	$0--$	$0--$	$0--$	$---$	$1\circ\circ$	$---$	$---$	$a \vee$
s_{4_1}	$---$	$-0-$	$-0-$	$---$	$\circ 1\circ$	$-0-$	$-0-$	$\neg b \vee$
s_{4_2}	$---$	$--0$	$---$	$--0$	$\circ\circ 1$	$--0$	$---$	c
s_{5_0}	$0--$	$0--$	$0--$	$---$	$---$	$1\circ\circ$	$---$	$a \vee$
s_{5_1}	$-0-$	$---$	$---$	$-0-$	$-0-$	$\circ 1\circ$	$---$	$b \vee$
s_{5_2}	$--0$	$---$	$--0$	$---$	$--0$	$\circ\circ 1$	$--0$	$\neg c$
s_{6_0}	$0--$	$0--$	$0--$	$---$	$---$	$---$	$1\circ\circ$	$a \vee$
s_{6_1}	$-0-$	$---$	$---$	$-0-$	$-0-$	$---$	$\circ 1\circ$	$b \vee$
s_{6_2}	$---$	$--0$	$---$	$--0$	$---$	$--0$	$\circ\circ 1$	c

For a better overview, a status row, representing the selection status of the entire problem is introduced. It is labeled P and behaves essentially like a matrix row. If an entire column of the satoku matrix is filled with zeroes, the corresponding cell in the status line becomes 0:

P	− − −	− − −	− − −	− − −	− − −	− − −	− − −	
s_{0_0}	$1\circ\circ$	− − −	− − −	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{0_1}	$\circ 1\circ$	$-0-$	$-0-$	− − −	− − −	$-0-$	$-0-$	$\neg b \vee$
s_{0_2}	$\circ\circ 1$	$--0$	− − −	$--0$	− − −	$--0$	− − −	c
s_{1_0}	− − −	$1\circ\circ$	− − −	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{1_1}	$-0-$	$\circ 1\circ$	− − −	$-0-$	$-0-$	− − −	− − −	$b \vee$
s_{1_2}	$--0$	$\circ\circ 1$	$--0$	− − −	$--0$	− − −	$--0$	$\neg c$
s_{2_0}	− − −	− − −	$1\circ\circ$	$0--$	$0--$	$0--$	$0--$	$\neg a \vee$
s_{2_1}	$-0-$	− − −	$\circ 1\circ$	$-0-$	$-0-$	− − −	− − −	$b \vee$
s_{2_2}	− − −	$--0$	$\circ\circ 1$	$--0$	− − −	$--0$	− − −	c
s_{3_0}	$0--$	$0--$	$0--$	$1\circ\circ$	− − −	− − −	− − −	$a \vee$
s_{3_1}	− − −	$-0-$	$-0-$	$\circ 1\circ$	− − −	$-0-$	$-0-$	$\neg b \vee$
s_{3_2}	$--0$	− − −	$--0$	$\circ\circ 1$	$--0$	− − −	$--0$	$\neg c$
s_{4_0}	$0--$	$0--$	$0--$	− − −	$1\circ\circ$	− − −	− − −	$a \vee$
s_{4_1}	− − −	$-0-$	$-0-$	− − −	$\circ 1\circ$	$-0-$	$-0-$	$\neg b \vee$
s_{4_2}	− − −	$--0$	− − −	$--0$	$\circ\circ 1$	$--0$	− − −	c
s_{5_0}	$0--$	$0--$	$0--$	− − −	− − −	$1\circ\circ$	− − −	$a \vee$
s_{5_1}	$-0-$	− − −	− − −	$-0-$	$-0-$	$\circ 1\circ$	− − −	$b \vee$
s_{5_2}	$--0$	− − −	$--0$	− − −	$--0$	$\circ\circ 1$	$--0$	$\neg c$
s_{6_0}	$0--$	$0--$	$0--$	− − −	− − −	− − −	$1\circ\circ$	$a \vee$
s_{6_1}	$-0-$	− − −	− − −	$-0-$	$-0-$	− − −	$\circ 1\circ$	$b \vee$
s_{6_2}	− − −	$--0$	− − −	$--0$	− − −	$--0$	$\circ\circ 1$	c

Finally, the original SAT problem is augmented with some tautologies to represent the (possible) selection of either a variable or its negation:

$$
\begin{array}{l}
(\neg \mathtt{a} \vee \neg \mathtt{b} \vee \mathtt{c}) \wedge \\
(\neg \mathtt{a} \vee \mathtt{b} \vee \neg \mathtt{c}) \wedge \\
(\neg \mathtt{a} \vee \mathtt{b} \vee \mathtt{c}) \wedge \\
(\mathtt{a} \vee \neg \mathtt{b} \vee \neg \mathtt{c}) \wedge \\
(\mathtt{a} \vee \neg \mathtt{b} \vee \mathtt{c}) \wedge \\
(\mathtt{a} \vee \mathtt{b} \vee \neg \mathtt{c}) \wedge \\
(\mathtt{a} \vee \mathtt{b} \vee \mathtt{c}) \wedge \\
(\mathtt{a} \vee \neg \mathtt{a}) \wedge \\
(\mathtt{b} \vee \neg \mathtt{b}) \wedge \\
(\mathtt{c} \vee \neg \mathtt{c})
\end{array}
$$

Proof: For all propositional formulas P the following statement is true:

$$
\begin{array}{lll|l}
 & P \wedge (p \vee \neg p) & & p \vee \neg p = \mathtt{T} \\
= & P \wedge \mathtt{T} & & p \wedge \mathtt{T} = p \\
= & P & &
\end{array}
$$

Since these tautologies are entirely optional, they are separated from the original core problem (upper left matrix region) by a double line.

Note, that so far nothing has happened to the regular inverted adjacency matrix. The information is exactly the same, only the visual representation is different.

P	− − −	− − −	− − −	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− − −	− − −	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{0_1}	∘ 1 ∘	− 0 −	− 0 −	− − −	− − −	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{0_2}	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− − 0	− − −	− −	− −	1 0	c
s_{1_0}	− − −	1 ∘ ∘	− − −	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{1_1}	− 0 −	∘ 1 ∘	− − −	− 0 −	− 0 −	− − −	− − −	− −	1 0	− −	$b \vee$
s_{1_2}	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− − 0	− −	− −	0 1	$\neg c$
s_{2_0}	− − −	− − −	1 ∘ ∘	0 − −	0 − −	0 − −	0 − −	0 1	− −	− −	$\neg a \vee$
s_{2_1}	− 0 −	− − −	∘ 1 ∘	− 0 −	− 0 −	− − −	− − −	− −	1 0	− −	$b \vee$
s_{2_2}	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− − −	− −	− −	1 0	c
s_{3_0}	0 − −	0 − −	0 − −	1 ∘ ∘	− − −	− − −	− − −	1 0	− −	− −	$a \vee$
s_{3_1}	− − −	− 0 −	− 0 −	∘ 1 ∘	− − −	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{3_2}	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− − 0	− −	− −	0 1	$\neg c$
s_{4_0}	0 − −	0 − −	0 − −	− − −	1 ∘ ∘	− − −	− − −	1 0	− −	− −	$a \vee$
s_{4_1}	− − −	− 0 −	− 0 −	− − −	∘ 1 ∘	− 0 −	− 0 −	− −	0 1	− −	$\neg b \vee$
s_{4_2}	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− − −	− −	− −	1 0	c
s_{5_0}	0 − −	0 − −	0 − −	− − −	− − −	1 ∘ ∘	− − −	1 0	− −	− −	$a \vee$
s_{5_1}	− 0 −	− − −	− − −	− 0 −	− 0 −	∘ 1 ∘	− − −	− −	1 0	− −	$b \vee$
s_{5_2}	− − 0	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− − 0	− −	− −	0 1	$\neg c$
s_{6_0}	0 − −	0 − −	0 − −	− − −	− − −	− − −	1 ∘ ∘	1 0	− −	− −	$a \vee$
s_{6_1}	− 0 −	− − −	− − −	− 0 −	− 0 −	− − −	∘ 1 ∘	− −	1 0	− −	$b \vee$
s_{6_2}	− − −	− − 0	− − −	− − 0	− − −	− − 0	∘ ∘ 1	− −	− −	1 0	c
s_{7_0}	0 − −	0 − −	0 − −	− − −	− − −	− − −	− − −	1 ∘	− −	− −	a
s_{7_1}	− − −	− − −	− − −	0 − −	0 − −	0 − −	0 − −	∘ 1	− −	− −	$\neg a$
s_{8_0}	− 0 −	− − −	− − −	− 0 −	− 0 −	− − −	− − −	− −	1 ∘	− −	b
s_{8_1}	− − −	− 0 −	− 0 −	− − −	− − −	− 0 −	− 0 −	− −	∘ 1	− −	$\neg b$
s_{9_0}	− − −	− − 0	− − −	− − 0	− − −	− − 0	− − −	− −	− −	1 ∘	c
s_{9_1}	− − 0	− − −	− − 0	− − −	− − 0	− − −	− − 0	− −	− −	∘ 1	$\neg c$