Combinatorial Decision Making and Optimization (Module 1)

Last update: 11 April 2024

Academic Year 2023 – 2024
Alma Mater Studiorum · University of Bologna

Contents

1 Constraint programming **1**
1.1 Definitions . . . 1
1.2 Modeling techniques . . . 1
1.3 Constraints . . . 2
1.3.1 Local consistency . . . 2
1.3.2 Constraint propagation . . . 3
1.3.3 Global constraints . . . 3
1.4 Search . . . 5
1.4.1 Search heuristics . . . 5
1.4.2 Constraint optimization problems . . . 6
1.4.3 Large neighborhood search (LNS) . . . 6

2 SAT solver **7**
2.1 Solvers . . . 7
2.1.1 Resolution . . . 7
2.1.2 DPLL algorithm . . . 8
2.1.3 Conflict-driven clause learning (CDCL) . . . 9
2.2 Encodings . . . 12

1 Constraint programming

1.1 Definitions

Constraint satisfaction problem (CSP) Triple $\langle X, D, C \rangle$ where: Constraint satisfaction problem (CSP)

- X is the set of decision variables $\{x_1, \ldots, x_n\}$.
- D is the set of domains $\{D(X_1), \ldots, D(X_n)\}$ or $\{D_1, \ldots, D_n\}$ for X.
- C is the set of constraints $\{C_1, \ldots, C_m\}$. Each C_i is a relation over the domain of X (i.e. $C_i \subseteq D_j \times \cdots \times D_k$).

Constraint optimization problem (COP) Tuple $\langle X, D, C, f \rangle$ where $\langle X, D, C \rangle$ is a CSP and f is the objective variable to minimize or maximize. Constraint optimization problem (COP)

Constraint Constraint

Extensional representation List all allowed combinations.

Intensional representation Declarative relations between variables.

Symmetry Search states that lead to the same result.

Variable symmetry A permutation of the assignment order of the variables results in the same feasible or unfeasible solution. Variable symmetry

Value symmetry A permutation of the values in the domain results in the same feasible or unfeasible solution. Value symmetry

> **Remark.** Variable and value symmetries can be combined resulting in a total of $2n!$ possible symmetries.

1.2 Modeling techniques

Auxiliary variables Add new variables to capture constraints difficult to model or to reduce the search space by collapsing multiple variables into one. Auxiliary variables

Global constraints Relation between an arbitrary number of variables. It is usually computationally faster than listing multiple constraints. Global constraints

Implied constraints Semantically redundant constraints with the advantage of pruning the search space earlier. Implied constraints

> **Remark.** A purely redundant constraint is also an implied constraint but it does not give any computational improvement.

Symmetry breaking constraints Constraints to avoid considering symmetric states. Usually, it is sufficient to fix an ordering of the variables. Symmetry breaking constraints

> **Remark.** When introducing symmetry breaking constraints, it might be possible to add new simplifications and implied constraints.

Dual viewpoint Modeling a problem from a different perspective might result in a more efficient search. Dual viewpoint

Example. Exploiting geometric symmetries.

Combined model Merging two or more models of the same problem by adding channeling constraints to guarantee consistency. Combined model

Combining two models can be useful for obtaining the advantages of both (e.g. one model uses global constraints, while the other handles symmetries).

Remark. When combining multiple models, some constraints might be simplified as one of the models already captures it natively.

1.3 Constraints

1.3.1 Local consistency

Examine individual constraints and detect inconsistent partial assignments.

Generalized arc consistency (GAC)

Generalized arc consistency (GAC)

Support Given a constraint defined on k variables $C \subseteq D(X_1) \times \cdots \times D(X_k)$, each tuple $(d_1, \ldots, d_k) \in C$ (i.e. allowed variables assignment) is a support for C.

A constraint $C(X_1, \ldots, X_k)$ is GAC (GAC(C)) iff:

$$\forall X_i \in \{X_1, \ldots, X_k\}, \forall v \in D(X_i) : v \text{ belongs to a support for } C$$

Remark. A CSP is GAC when all its constraints are GAC.

Example (Generalized arc consistency)**.** Given the variables $D(X_1) = \{1, 2, 3\}$, $D(X_2) = \{1, 2\}$, $D(X_3) = \{1, 2\}$ and the constraint C : `alldifferent`$([X_1, X_2, X_3])$, C is not GAC as $1 \in D(X_1)$ and $2 \in D(X_1)$ do not have a support.
By applying a constraint propagation algorithm, we can reduce the domain to: $D(X_1) = \{\cancel{1}, \cancel{2}, 3\}$ and $D(X_2) = D(X_3) = \{1, 2\}$. Now C is GAC.

Arc consistency (AC) A constraint is arc consistent when its binary constrains are GAC. Arc consistency (AC)

Example (Arc consistency)**.** Given the variables $D(X_1) = \{1, 2, 3\}$, $D(X_2) = \{2, 3, 4\}$ and the constraint $C : X_1 = X_2$, C is not arc consistent as $1 \in D(X_1)$ and $4 \in D(X_2)$ do not have a support.
By applying a constraint propagation algorithm, we can reduce the domain to: $D(X_1) = \{\cancel{1}, 2, 3\}$ and $D(X_2) = \{2, 3, \cancel{4}\}$. Now C is arc consistent.

Bounds consistency (BC) Can be applied on totally ordered domains. The domain of a variable X_i is relaxed from $D(X_i)$ to the interval $[\min\{D(X_i)\}, \max\{D(X_i)\}]$. Bounds consistency (BC)

Bound support Given a constraint defined on k variables $C(X_1, \ldots, X_k)$, each tuple $(d_1, \ldots, d_k) \in C$, where $d_i \in [\min\{D(X_i)\}, \max\{D(X_i)\}]$, is a bound support for C.

A constraint $C(X_1, \ldots, X_k)$ is BC (BC(C)) iff:

$$\forall X_i \in \{X_1, \ldots, X_k\} :$$
$$\min\{D(X_i)\} \text{ and } \max\{D(X_i)\} \text{ belong to a bound support for } C$$

Remark. BC might not detect all GAC inconsistencies, but it is computationally cheaper.

Remark. On monotonic constraints, BC and GAC are equivalent.

Example. Given the variables $D(X_1) = D(X_2) = D(X_3) = \{1, 3\}$ and the constraint C : `alldifferent`$([X_1, X_2, X_3])$, C is BC as all $\min\{D(X_i)\}$ and $\max\{D(X_i)\}$ belong to the bound support $\{(d_1, d_2, d_3) \mid d_i \in [1, 3] \land d_1 \neq d_2 \neq d_3\}$ On the other hand, C fails with GAC.

1.3.2 Constraint propagation

Constraint propagation Algorithm that removes values from the domain of the variables to achieve a given level of consistency. Constraint propagation

Constraint propagation algorithms interact with each other and already propagated constraints might be woke up again by another constraint. Propagation will eventually reach a fixed point.

Specialized propagation Propagation algorithm specific to a given constraint. Allows to exploit the semantics of the constraint for a generally more efficient approach. Specialized propagation

1.3.3 Global constraints

Constraints to capture complex, non-binary, and recurring features of the variables. Usually, global constraints are enforced using specialized propagation algorithms.

Counting constraints

Constrains the number of variables satisfying a condition or the occurrences of certain values. Counting constraints

All-different Enforces that all variables assume a different value.

$$\texttt{alldifferent}([X_1, \ldots, X_k]) \iff \forall i, j, \in \{1, \ldots, k\}, i \neq j : X_i \neq X_j$$

Global cardinality Enforces the number of times some values should appear among the variables.

$$\texttt{gcc}([X_1, \ldots, X_k],[v_1, \ldots, v_m], [O_1, \ldots, O_m]) \iff \forall j \in \{1, \ldots, m\} : |\{X_i \mid X_i = v_j\}| = O_j$$

Among Constrains the number of occurrences of certain values among the variables.

$$\texttt{among}([X_1, \ldots, X_k], \{v_1, \ldots, v_n\}, l, u) \iff l \leq |\{X_i \mid X_i \in \{v_1, \ldots, v_n\}\}| \leq u$$

Sequencing constraints

Enforces a pattern on a sequence of variables. Sequencing constraints

Sequence Enforces the number of times certain values can appear in a subsequence of a given length q.

$$\texttt{sequence}(l, u, q,[X_1, \ldots, X_k], \{v_1, \ldots, v_n\}) \iff \forall i \in [1, \ldots, k - q + 1] : \texttt{among}([X_1, \ldots, X_{i+q-1}], \{v_1, \ldots, v_n\}, l, u)$$

Scheduling constraints

Useful to schedule tasks with release times, duration, deadlines, and resource limitations.

Scheduling constraints

Disjunctive resource Enforces that the tasks do not overlap over time. Given the start time S_i and the duration D_i of k tasks:

$$\texttt{disjunctive}([S_1,\ldots,S_k],[D_1,\ldots,D_k]) \iff \forall i<j : (S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$$

Cumulative resource Constrains the usage of a shared resource. Given a resource with capacity C and the start time S_i, the duration D_i, and the resource requirement R_i of k tasks:

$$\texttt{cumulative}([S_1,\ldots,S_k],[D_1,\ldots,D_k],[R_1,\ldots,R_k],C) \iff \forall u \in \{D_1,\ldots,D_k\} : \sum_{i \text{ s.t. } S_i \leq u < S_i + D_i} R_i \leq C$$

Ordering constraints

Enforce an ordering between variables or values.

Ordering constraints

Lexicographic ordering Enforces that a sequence of variables is lexicographically less than or equal to another sequence.

$$\begin{aligned}\texttt{lex}{\leq}([X_1,\ldots,X_k],[Y_1,\ldots,Y_k]) \iff & X_1 \leq Y_1 \wedge \\ & (X_1 = Y_1 \Rightarrow X_2 \leq Y_2) \wedge \\ & \cdots \wedge \\ & ((X_1 = Y_1 \wedge \cdots \wedge X_{k-1} = Y_{k-1}) \Rightarrow X_k \leq Y_k)\end{aligned}$$

Generic purpose constraints

Define constraints in an extensive way.

Generic purpose constraints

Table Associate to the variables their allowed assignments.

Specialized propagation

Constraint decomposition A global constraint is decomposed into smaller and simpler constraints with known propagation algorithms.

Constraint decomposition

Remark. As the problem is decomposed, some inconsistencies may not be detected.

Example. (Decomposition of `among`) The `among` constraint can be decomposed as follows:

Variables B_i with $D(B_i) = \{0,1\}$ for $1 \leq i \leq k$.

Constraints

- $C_i : B_i = 1 \iff X_i \in v$ for $1 \leq i \leq k$.
- $C_{k+1} : \sum_i B_i = N$.

AC(C_i) and BC(C_{k+1}) ensures GAC on `among`.

Dedicated propagation algorithm Ad-hoc algorithm that implements an efficient propagation.

Dedicated propagation algorithm

Example. (`alldifferent` through maximal matching)

Bipartite graph Graph where the edges are partitioned in two groups U and V. Nodes in U can only connect to nodes in V.

Maximal matching Largest subsets of edges such that there are no edges with nodes in common.

Define a bipartite graph $G = (U \cup V, E)$ where:

- $U = \{X_1, \ldots, X_k\}$ are the variables.
- $V = D(X_1) \cup \cdots \cup D(X_k)$ are the possible values of the variables.
- $E = \{(X_i, v) \mid X_i \in U, v \in V : v \in D(X_i)\}$ contains the edges that connect every variable in U to its possible values in V.

All the possible variable assignments of $X_1, \ldots, X_k$ are the maximal matchings in G.

1.4 Search

Backtracking tree search Tree where nodes are variables and branches are variable assignments.

Backtracking tree search

Systematic search Instantiate and explore the tree depth first. Constraints are checked when all variables are assigned (i.e. when a leaf is reached) and the search backtracks of one decision if it fails.

This approach has exponential complexity.

Systematic search

Search and propagation Propagate the constraints after an assignment to remove inconsistent values from the domain of yet unassigned variables.

Search and propagation

1.4.1 Search heuristics

Static heuristic The order of exploration of the variables is fixed before search.

Static heuristic

Dynamic heuristic The order of exploration of the variables is determined during search.

Dynamic heuristic

Fail-first (FF) Try the variables that are most likely to fail in order to maximize propagation.

Fail-first (FF)

Minimum domain Assign the variable with the minimum domain size.

Minimum domain

Most constrained Assign the variable with the maximum degree (i.e. number of constraints that involve it).

Most constrained

Combination Combine both minimum domain and most constrained to minimize the value $\frac{\text{domain size}}{\text{degree}}$.

Combination

Weighted degree Each constraint C starts with a weight $w(C) = 1$. During propagation, when a constraint C fails, its weight is increased by 1.

Weighted degree

The weighted degree of a variable X_i is:

$$w(X_i) = \sum_{C \text{ s.t. } C \text{ involves } X_i} w(C)$$

Assign the variable with minimum $\frac{|D(X_i)|}{w(X_i)}$.

Heavy tail behavior Instances of a problem that are particularly hard and expensive to solve.

Randomization Sometimes, make a random choice: Randomization

- Randomly choose the variable or value to assign.
- Break ties randomly.

Restarting Restart the search after a certain amount of resources (e.g. search steps) have been consumed. The new search can exploit past knowledge, change heuristics, or use randomization. Restarting

Constant restart Restart after a fixed number L of resources have been used.

Geometric restart At each restart, the resource limit L is multiplied by a factor α. This will result in a sequence $L, \alpha L, \alpha^2 L, \dots$.

Luby restart

Luby sequence The first element of the sequence is 1. Then, the sequence is iteratively computed as follows:

- Repeat the current sequence.
- Add 2^{i+1} at the end of the sequence.

Example. $[1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \dots]$

At the i-th restart, the resource limit L is multiplied by a factor determined by the i-th element of the Luby sequence.

Remark. Weighted degree and restarts work well together when weights are carried over.

Remark. Restarting on deterministic heuristics does not give any advantage.

1.4.2 Constraint optimization problems

Branch and bound Solves a COP by solving a sequence of CSPs. Branch and bound

1. Find a feasible solution and add a bounding constraint to enforce that future solutions are better than this one.
2. Backtrack the last decision and look for a new solution on the same tree with the new constraint.
3. Repeat until the problem becomes unfeasible. The last solution is optimal.

1.4.3 Large neighborhood search (LNS)

Hybrid between constraint programming and heuristic search. Large neighborhood search (LNS)

1. Find an initial solution s using CP.
2. Create a partial solution $N(s)$ by taking some assignments from the solution s and leaving the other unassigned.
3. Explore the large neighborhood using CP starting from $N(s)$.

2 SAT solver

Satifiability (SAT) Given a propositional formula f, find an assignment of the variables to make the formula true. SAT

If f is satisfiable, we say that it is SAT. Otherwise, it is UNSAT.

Theorem 2.0.1. (Cook-Levin) SAT is NP-complete. Cook-Levin theorem

NP-complete A problem is NP-complete iff it is in NP and every NP problem can be reduced to it.

| **Remark.** All problems in NP can be reduced to SAT.

Satisfiable formula A formula f is satisfiable iff there is an assignment of the variables that makes it true. Satisfiable formula

Valid formula A formula f is valid iff any assignment of the variables makes it true. Valid formula

| **Remark.** f is valid iff $\neg f$ is UNSAT.

SAT solver Given a problem, the following steps should be followed to solve it using a SAT solver: SAT solver

1. Formalize the problem into a propositional formula.
2. Depending on the solver, it might be necessary to convert it into CNF.
3. Feed the formula to a SAT solver.
4. The result can be an assignment of the variables (SAT) or a failure (UNSAT).

2.1 Solvers

2.1.1 Resolution

Resolution Basic approach for SAT. Prove that a set of CNF clauses is UNSAT by deriving new clauses until a contradiction is reached. Resolution

Resolution rules to derive new clauses are:

Resolution rule $\dfrac{p \vee V \qquad \neg p \vee W}{V \vee W}$ Resolution rule

Unit resolution $\dfrac{p \qquad \neg p \vee W}{W}$ $\qquad$ $\dfrac{\neg p \qquad p \vee V}{V}$ Unit resolution

Note that this is equivalent to the rule of implication elimination:

$$\frac{p \qquad p \Rightarrow W}{W} \qquad\qquad \frac{\neg p \qquad \neg p \Rightarrow V}{V}$$

Example. Given the CNF formula:

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

Resolution can be applied as follows:

- The starting clauses are: $(p \vee q) \cdot (\neg r \vee s) \cdot (\neg q \vee r) \cdot (\neg r \vee \neg s) \cdot (\neg p \vee r)$
- From $(p \vee q)$ and $(\neg q \vee r)$, one can derive $(p \vee r)$.
- From $(\neg p \vee r)$ and $(p \vee r)$, one can derive r.
- From $(\neg r \vee s)$ and r, one can derive s.
- From $(\neg r \vee \neg s)$ and s, one can derive $\neg r$.
- From r and $\neg r$, one reaches $\bot$.

2.1.2 DPLL algorithm

DPLL Algorithm based on unit resolution. Given a set of clauses, DPLL is applied as follows: DPLL

1. Apply unit resolution as long as possible (i.e. for every clause containing a single literal l, remove $\neg l$ from all clauses. Clauses containing l can also be removed as they are redundant).
2. Choose a variable p.
3. Recursively apply DPLL by assuming p in one call and $\neg p$ in another.

Theorem 2.1.1. DPLL is complete.

Remark. DPLL efficiency strongly depends on the variables.

Example. Given a CNF formula with clauses:

$$(\neg p \vee \neg s) \cdot (\neg p \vee \neg r) \cdot (\neg q \vee \neg t) \cdot (p \vee r) \cdot (p \vee s) \cdot (r \vee t) \cdot (\neg s \vee t) \cdot (q \vee s)$$

There are no unit clauses, so we choose the variable p and consider the cases where p and where $\neg p$.
By assuming p, unit resolution can be applied as follows:

- p with $(\neg p \vee \neg s)$ derives $\neg s$.
- p with $(\neg p \vee \neg r)$ derives $\neg r$.
- $\neg s$ with $(q \vee s)$ derives q.
- $\neg r$ with $(r \vee t)$ derives t.
- q with $(\neg q \vee \neg t)$ derives $\neg t$.
- t and $\neg t$ bring to $\bot$. This branch is UNSAT.

By assuming $\neg p$, unit resolution can be applied as follows:

- $\neg p$ with $(p \vee r)$ derives r.
- $\neg p$ with $(p \vee s)$ derives s.
- s with $(\neg s \vee t)$ derives t.
- t with $(\neg q \vee \neg t)$ derives $\neg q$.

We reached a full assignment $\langle \neg p, r, s, t, \neg q \rangle$. The formula is SAT.

DPLL implementation An efficient implementation of DPLL uses a list M of the literals that have been decided or derived. DPLL algorithm

Notations

- A literal l holds in M ($M \models l$) iff l is in M.
- M contradicts a clause C ($M \models \neg C$) iff for every literal l in C, $M \models \neg l$.
- A literal l is undefined iff $M \not\models l$ and $M \not\models \neg l$.
- A decision literal (i.e. literal assumed to hold) is denoted as $l^{\mathcal{D}}$.

The algorithm follows four rules:

`UnitPropagate` If the CNF contains a clause $(A \vee l)$ such that $M \models \neg A$ and l is an undefined literal, then there is a transition:

$$M \to Ml$$

`Decide` When `UnitPropagate` it not possible, an undefined literal l is assumed to hold (it becomes a decision literal):

$$M \to Ml^{\mathcal{D}}$$

`Backtrack` When a branch is UNSAT, the algorithm backtracks to the nearest decision literal and negates it. Given a clause C of the CNF, if $Ml^{\mathcal{D}}N \models \neg C$ and N does not contain other decision literals, then there is a transition:

$$Ml^{\mathcal{D}}N \to M\neg l$$

`Fail` Given a clause C of the CNF, if $M \models \neg C$ and M contains a definitive assignment of all the variables (i.e. no decision literals), then there is a transition:

$$M \to \text{fail}$$

In the end, if the formula is SAT, M contains an assignment of the variables.

Remark. Compared to resolution, DPLL is more memory efficient.

2.1.3 Conflict-driven clause learning (CDCL)

DPLL with two improvements:

Clause learning When a contradiction is found, augment the original CNF with a conflict clause to prune the search space.

The conflict clause is obtained through resolution by combining the two clauses that cause the contradiction.

`Backjumping` Depending on the type of contradiction, skip decision literals that are not causing the conflict and directly backtrack to an earlier relevant decision literal.

Given a clause C of the CNF, if $Ml^{\mathcal{D}}N \models \neg C$ and there is a clause $(C' \vee l')$ derivable from the CNF such that $M \models \neg C'$ and l' is undefined, then there is a transition:

$$Ml^{\mathcal{D}}N \to Ml'$$

Note that N might contain other decision literals and $(C' \vee l')$ is a conflict clause.

Remark. `Backjumping` can be seen as `UnitPropagate` applied on conflict clauses.

Implication graph DAG $G = (V, E)$ to record the history of decisions and unit resolutions.

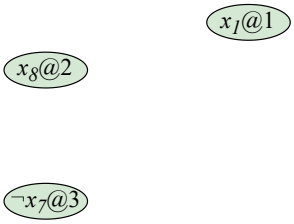
Vertex Can be:

- A decision or derived literal (denoted as $l@d$, where l is the literal and d the decision level it originated from).
- A marker to indicate a conflict (denoted as $\kappa@d$, where d is the decision level it originated from).

Edge Has form $v \xrightarrow{c_i} w$ and indicates that w is obtained using the clause c_i and the literal v.

Example. Starting from the clauses $c_1, \ldots, c_6$ of a CNF, one can build the implication graph while running DPLL/CDCL.
In this example, the decision literals (green nodes) are assigned following this order: $x_1 \to x_8 \to \neg x_7$:

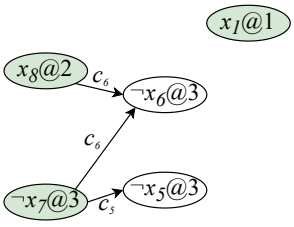
c_1: $\neg x_1 \vee x_2 \vee \neg x_4$
c_2: $\neg x_1 \vee \neg x_2 \vee x_3$
c_3: $\neg x_3 \vee \neg x_4$
c_4: $x_4 \vee x_5 \vee x_6$
c_5: $\neg x_5 \vee x_7$
c_6: $\neg x_6 \vee x_7 \vee \neg x_8$



$$M \to M x_1^{\mathcal{D}} x_8^{\mathcal{D}} \neg x_7^{\mathcal{D}}$$

Then, it is possible to derive implied variables using unit resolution starting from the decision literals:

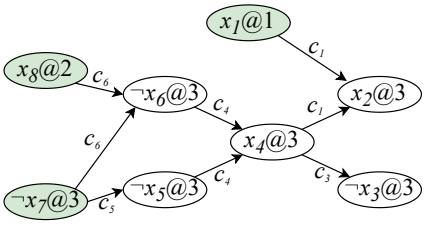
c_1: $\neg x_1 \vee x_2 \vee \neg x_4$
c_2: $\neg x_1 \vee \neg x_2 \vee x_3$
c_3: $\neg x_3 \vee \neg x_4$
c_4: $x_4 \vee x_5 \vee x_6$
c_5: $\neg x_5 \vee x_7$
c_6: $\neg x_6 \vee x_7 \vee \neg x_8$



$$M \to M \neg x_6 \neg x_5$$

And so on:

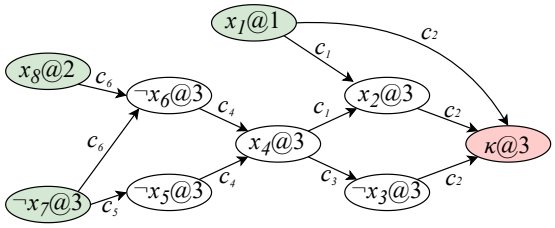
c_1: $\neg x_1 \vee x_2 \vee \neg x_4$
c_2: $\neg x_1 \vee \neg x_2 \vee x_3$
c_3: $\neg x_3 \vee \neg x_4$
c_4: $x_4 \vee x_5 \vee x_6$
c_5: $\neg x_5 \vee x_7$
c_6: $\neg x_6 \vee x_7 \vee \neg x_8$



$$M \to M x_4 x_2 \neg x_3$$

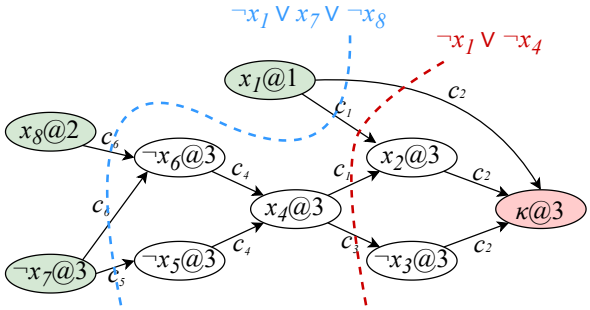
As c_2 is UNSAT, we reached a contradiction:

c_1: $\neg x_1 \vee x_2 \vee \neg x_4$
c_2: $\neg x_1 \vee \neg x_2 \vee x_3$
c_3: $\neg x_3 \vee \neg x_4$
c_4: $x_4 \vee x_5 \vee x_6$
c_5: $\neg x_5 \vee x_7$
c_6: $\neg x_6 \vee x_7 \vee \neg x_8$



Remark. Any cut that separates sources (decision literals) and the sink (contradiction node) is a valid conflict clause for backjumping.

Example.



Unit implication point (UIP) Given an implication graph where the latest decision node is $(l@d)$ and the conflict node is $(\kappa@d)$, a node is a unit implication point iff it appears in all the paths from $(l@d)$ to $(\kappa@d)$.

The UIP closest to the conflict is denoted as 1UIP.

Remark. The decision node $(l@d)$ itself is a UIP.

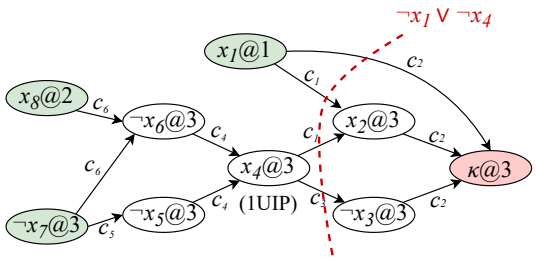
1UIP-based backjumping In case of conflict, use the conflict clause obtained by cutting in front of the 1UIP.

Remark. This allows to:

- Reduce the size of the conflict clause.
- Guarantee the highest backtrack jump.

Example.

c_1: $\neg x_1 \vee x_2 \vee \neg x_4$
c_2: $\neg x_1 \vee \neg x_2 \vee x_3$
c_3: $\neg x_3 \vee \neg x_4$
c_4: $x_4 \vee x_5 \vee x_6$
c_5: $\neg x_5 \vee x_7$
c_6: $\neg x_6 \vee x_7 \vee \neg x_8$
c: $\neg x_1 \vee \neg x_4$



By adding the conflict clause $(\neg x_1 \vee \neg x_4)$, we can backjump up to $x_8^{\mathcal{D}}$ (not to $x_1^{\mathcal{D}}$ as it makes the conflict clause a unit).

$$x_1^{\mathcal{D}} x_8^{\mathcal{D}} \neg x_7^{\mathcal{D}} \neg x_6 \neg x_5 x_4 x_2 \neg x_3 \rightarrow x_1^{\mathcal{D}} \neg x_4$$

2.2 Encodings

Cardinality constraints Constraints of the form: Cardinality constraints

$$\sum_{1 \leq i \leq n} x_i \bowtie k$$

where $k \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \neq, \geq, >\}$.

$\mathbf{k = 1}$ constraints Constraints of the form: $k = 1$ constraints

$$\sum_{1 \leq i \leq n} x_i \bowtie 1$$

`ExactlyOne` `ExactlyOne`

$$\texttt{ExactlyOne}([x_1, \ldots, x_n]) \iff \texttt{AtMostOne}([x_1, \ldots, x_n]) \land \texttt{AtLeastOne}([x_1, \ldots, x_n])$$

`AtLeastOne` `AtLeastOne`

$$\texttt{AtLeastOne}([x_1, \ldots, x_n]) \iff x_1 \lor x_2 \lor \cdots \lor x_n$$

`AtMostOne` There are different encoding approaches: `AtMostOne`

Pairwise encoding Constrains the fact that any pair of variables cannot be both true: `AtMostOne` pairwise encoding

$$\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff \bigwedge_{1 \leq i < j \leq n} \neg(x_i \land x_j)$$

This encoding does not require auxiliary variables and introduces $O(n^2)$ new clauses.

Sequential encoding Introduces n new variables s_i such that: `AtMostOne` sequential encoding

- If $s_i = 1$ and $s_{i-1} = 0$, then $x_i = 1$.
- If $s_i = 1$ and $s_{i-1} = 1$, then $x_i = 0$.

$$\begin{aligned}\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff & (x_1 \Rightarrow s_1) \\ & \land \bigwedge_{1 < i < n} \big[\big((x_i \lor s_{i-1}) \Rightarrow s_i\big) \land (s_{i-1} \Rightarrow \neg x_i)\big] \\ & \land (s_{n-1} \Rightarrow \neg x_n)\end{aligned}$$

By expanding the implications, it becomes:

$$\begin{aligned}\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff & (\neg x_1 \lor s_1) \\ & \land \bigwedge_{1 < i < n} [(\neg x_i \lor s_i) \land (\neg s_{i-1} \lor s_i) \land (\neg s_{i-1} \lor \neg x_i)] \\ & \land (\neg s_{n-1} \lor \neg x_n)\end{aligned}$$

This encoding requires $O(n)$ auxiliary variables and $O(n)$ new clauses.

Bitwise encoding Introduces $m = \log_2 n$ new variables r_i such that if the variable x_i is set to true, then $r_1, \ldots, r_m$ contain the binary encoding of the index $i - 1$. `AtMostOne` bitwise encoding

Let $b_{i,1}, \ldots, b_{i,m}$ be the m bits of the binary representation of i, the encoding is the following:

$$\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff \bigwedge_{1 \leq i \leq n} \big(x_i \Rightarrow (r_1 = b_{i-1,1} \wedge \cdots \wedge r_m = b_{i-1,m})\big)$$

By expanding the implications, it becomes:

$$\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff \bigwedge_{1 \leq i \leq n} \bigwedge_{1 \leq j \leq m} \big(\neg x_i \vee (r_j = b_{i-1,j})\big)$$

This encoding requires $O(\log_2 n)$ auxiliary variables and $O(n \log_2 n)$ new clauses.

Heule encoding Defines the encoding recursively: `AtMostOne` Heule encoding

- When $n \leq 4$, apply the pairwise encoding (which requires 6 clauses):

$$\texttt{AtMostOne}([x_1, \ldots, x_4]) \iff \bigwedge_{1 \leq i < j \leq n} \neg(x_i \wedge x_j)$$

- When $n > 4$, introduce a new variable y:

$$\texttt{AtMostOne}([x_1, \ldots, x_n]) \iff \texttt{AtMostOne}([x_1, x_2, x_3, y]) \wedge \texttt{AtMostOne}([\neg y, x_4, \ldots, x_n])$$

This encoding requires $O(n)$ auxiliary variables and $O(n)$ new clauses.

$\mathbf{k \in \mathbb{N}}$ constraints Constraints of the form: $k \in \mathbb{N}$ constraints

$$\sum_{1 \leq i \leq n} x_i \bowtie k \text{ where } k \in \mathbb{N}$$

Theorem 2.2.1. All the cardinality constraints can be expressed using `AtMostK` (i.e. $\leq$ relationship).

Proof.

$$\sum_{1 \leq i \leq n} x_i = k \iff \sum_{1 \leq i \leq n} (x_i \geq k) \wedge \sum_{1 \leq i \leq n} (x_i \leq k)$$
$$\sum_{1 \leq i \leq n} x_i \neq k \iff \sum_{1 \leq i \leq n} (x_i < k) \vee \sum_{1 \leq i \leq n} (x_i > k)$$
$$\sum_{1 \leq i \leq n} x_i \geq k \iff \sum_{1 \leq i \leq n} (\neg x_i \leq n - k)$$
$$\sum_{1 \leq i \leq n} x_i > k \iff \sum_{1 \leq i \leq n} (\neg x_i \leq n - k - 1)$$
$$\sum_{1 \leq i \leq n} x_i < k \iff \sum_{1 \leq i \leq n} (x_i \leq k - 1)$$

□

`AtMostK` There are different encoding approaches:

`AtMostK`

Generalized pairwise encoding Constrains the fact that any pair of $(k+1)$ variables cannot be all true:

`AtMostK` generalized pairwise encoding

$$\texttt{AtMostK}([x_1,\ldots,x_n],k) \iff \bigwedge_{\substack{M\subseteq\{1,\ldots,n\}\\|M|=k+1}} \bigvee_{i\in M} \neg x_i$$

This encoding does not require auxiliary variables and introduces $O(n^{k+1})$ new clauses.

Sequential encoding Introduces $n \cdot k$ new variables $s_{i,j}$ such that:

`AtMostK` sequential encoding

- If $s_{i,j} = 1$ and $s_{i-1,j} = 0$, then x_i is the j-th variable assigned to true.
- If $s_{i,j} = 1$ and $s_{i-1,j} = 1$, then $x_i = 0$ and there are j variables assigned to true among $x_1, \ldots, x_{i-1}$.

$$\begin{aligned}
&\texttt{AtMostK}([x_1,\ldots,x_n],k) \iff \\
&\qquad (x_1 \Rightarrow s_{1,1}) \wedge \bigwedge_{2\le j\le k} \neg s_{1,j} \\
&\qquad \wedge \bigwedge_{1<i<n} \left[\begin{array}{l}
\big((x_i \vee s_{i-1,1}) \Rightarrow s_{i,1}\big) \\
\wedge \bigwedge_{2\le j\le k} \Big(\big((x_i \wedge s_{i-1,j-1}) \vee s_{i-1,j}\big) \Rightarrow s_{i,j}\Big) \\
\wedge\, (s_{i-1,k} \Rightarrow \neg x_i)
\end{array}\right] \\
&\qquad \wedge\, (s_{n-1,k} \Rightarrow \neg x_n)
\end{aligned}$$

This encoding requires $O(nk)$ auxiliary variables and $O(nk)$ new clauses.

Arc-consistency Given a SAT encoding E of a normal constraint C, E is arc-consistent if:

Arc-consistency

- Whenever a partial assignment is inconsistent in C, unit resolution in E causes a contradiction.
- Otherwise, unit resolution in E discards arc-inconsistent values.

`<end of course>`