Difference Logic

Satisfiability Checking Seminar

Alex Ryndin
Supervisor: Gereon Kremer

WS 2016/2017

Outline

- Main Literature
- Difference Logic
- Example Problem: Job Scheduling
- SAT Checking
- Constraint Graph And Negative Cycles
- Conclusion

Main Literature

- [Cotton et al. 2004] Scott Cotton, Eugene Asarin, Oded Maler and Peter Niebert. **“Some progress in satisfiability checking for difference logic“**. In Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pages 263–276. Springer, 2004.
- [Cormen et al. 2009] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. **“Introduction to algorithms“**. MIT press, third edition, 2009.
  Note: the chapter 24 **“Single-Source Shortest Paths“** is relevant for the topic.

Difference Logic

- Difference logic – a special case of linear arithmetic logic,
- in which constraints have the following form:

$$x - y \prec c$$

  x, y – variables, c – constant and $\prec \in \{<, \leq\}$ – comparison operator.
- x, y, c – $\mathbb{Z}$ or $\mathbb{R}$.

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15))$$

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15))$$

SAT $p = True, q = False, u = 3, v = 0, w = 0$

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15))$$

SAT $p = True, q = False, u = 3, v = 0, w = 0$

$$\begin{aligned}\phi_2 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < 1) \\ & \wedge (y - z \leq -5) \wedge (y - v \leq 0)\end{aligned}$$

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15))$$

SAT $p = True, q = False, u = 3, v = 0, w = 0$

$$\begin{aligned} \phi_2 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < 1) \\ & \wedge (y - z \leq -5) \wedge (y - v \leq 0) \end{aligned}$$

SAT $u = 0, v = 3, w = 0, x = 3, y = 3, z = 8$

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \to (u - v < 3.3)) \wedge (q \to (v - w < -5.15))$$

SAT $p = \mathit{True}, q = \mathit{False}, u = 3, v = 0, w = 0$

$$\begin{aligned}\phi_2 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < 1) \\ & \wedge (y - z \leq -5) \wedge (y - v \leq 0)\end{aligned}$$

SAT $u = 0, v = 3, w = 0, x = 3, y = 3, z = 8$

$$\begin{aligned}\phi_3 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < -3) \\ & \wedge (y - z \leq -5) \wedge (y - w < 4)\end{aligned}$$

Difference Logic

A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15))$$

SAT $p = \mathit{True}, q = \mathit{False}, u = 3, v = 0, w = 0$

$$\begin{aligned}\phi_2 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < 1) \\ & \wedge (y - z \leq -5) \wedge (y - v \leq 0)\end{aligned}$$

SAT $u = 0, v = 3, w = 0, x = 3, y = 3, z = 8$

$$\begin{aligned}\phi_3 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < -3) \\ & \wedge (y - z \leq -5) \wedge (y - w < 4)\end{aligned}$$

UNSAT $(w - x \leq -3) \wedge (x - y < -3) \wedge (y - w < 4) \Rightarrow 0 < -2$

Difference Logic

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$
  zero – special pseudo-variable

Difference Logic

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$
  zero – special pseudo-variable
- $x \geq c \quad \Leftrightarrow \quad -x \leq -c \quad \Leftrightarrow \quad 0 - x \leq -c \quad \Leftrightarrow \quad zero - x \leq -c$

Difference Logic

- $x < c \;\; \Leftrightarrow \;\; x - 0 < c \;\; \Leftrightarrow \;\; x - zero < c$
  zero – special pseudo-variable
- $x \geq c \;\; \Leftrightarrow \;\; -x \leq -c \;\; \Leftrightarrow \;\; 0 - x \leq -c \;\; \Leftrightarrow \;\; zero - x \leq -c$
- $x \neq c \;\; \Leftrightarrow \;\; ((x < c) \vee (x > c))$
- $x = c \;\; \Leftrightarrow \;\; \neg((x < c) \vee (x > c))$

Difference Logic

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$
  $zero$ – special pseudo-variable
- $x \geq c \quad \Leftrightarrow \quad -x \leq -c \quad \Leftrightarrow \quad 0 - x \leq -c \quad \Leftrightarrow \quad zero - x \leq -c$
- $x \neq c \quad \Leftrightarrow \quad ((x < c) \vee (x > c))$
- $x = c \quad \Leftrightarrow \quad \neg((x < c) \vee (x > c))$
- An example:

$$(v = -3)$$

Difference Logic

- $x < c \;\;\Leftrightarrow\;\; x - 0 < c \;\;\Leftrightarrow\;\; x - zero < c$
  zero – special pseudo-variable
- $x \geq c \;\;\Leftrightarrow\;\; -x \leq -c \;\;\Leftrightarrow\;\; 0 - x \leq -c \;\;\Leftrightarrow\;\; zero - x \leq -c$
- $x \neq c \;\;\Leftrightarrow\;\; ((x < c) \vee (x > c))$
- $x = c \;\;\Leftrightarrow\;\; \neg((x < c) \vee (x > c))$
- An example:

$$
\begin{aligned}
& (v = -3) \\
\Leftrightarrow \;\; & (\neg((v < -3) \vee (v > -3)))
\end{aligned}
$$

Difference Logic

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$
  $zero$ – special pseudo-variable
- $x \geq c \quad \Leftrightarrow \quad -x \leq -c \quad \Leftrightarrow \quad 0 - x \leq -c \quad \Leftrightarrow \quad zero - x \leq -c$
- $x \neq c \quad \Leftrightarrow \quad ((x < c) \vee (x > c))$
- $x = c \quad \Leftrightarrow \quad \neg((x < c) \vee (x > c))$
- An example:

$$
\begin{aligned}
& (v = -3) \\
\Leftrightarrow \quad & (\neg((v < -3) \vee (v > -3))) \\
\Leftrightarrow \quad & (\neg((v < -3) \vee (-v < 3)))
\end{aligned}
$$

Difference Logic

- $x < c \;\Leftrightarrow\; x - 0 < c \;\Leftrightarrow\; x - zero < c$
  zero – special pseudo-variable
- $x \geq c \;\Leftrightarrow\; -x \leq -c \;\Leftrightarrow\; 0 - x \leq -c \;\Leftrightarrow\; zero - x \leq -c$
- $x \neq c \;\Leftrightarrow\; ((x < c) \vee (x > c))$
- $x = c \;\Leftrightarrow\; \neg((x < c) \vee (x > c))$
- An example:

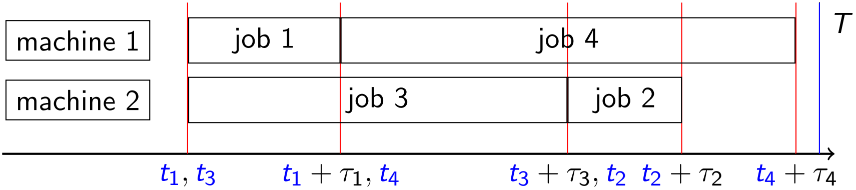
$$\begin{aligned} & (v = -3) \\ \Leftrightarrow \;\; & (\neg((v < -3) \vee (v > -3))) \\ \Leftrightarrow \;\; & (\neg((v < -3) \vee (-v < 3))) \\ \Leftrightarrow \;\; & (\neg((v - 0 < -3) \vee (0 - v < 3))) \end{aligned}$$

Difference Logic

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$
  zero – special pseudo-variable
- $x \geq c \quad \Leftrightarrow \quad -x \leq -c \quad \Leftrightarrow \quad 0 - x \leq -c \quad \Leftrightarrow \quad zero - x \leq -c$
- $x \neq c \quad \Leftrightarrow \quad ((x < c) \vee (x > c))$
- $x = c \quad \Leftrightarrow \quad \neg((x < c) \vee (x > c))$
- An example:

$$
\begin{aligned}
& (v = -3) \\
\Leftrightarrow \quad & (\neg((v < -3) \vee (v > -3))) \\
\Leftrightarrow \quad & (\neg((v < -3) \vee (-v < 3))) \\
\Leftrightarrow \quad & (\neg((v - 0 < -3) \vee (0 - v < 3))) \\
\Leftrightarrow \quad & (\neg((v - zero < -3) \vee (zero - v < 3)))
\end{aligned}
$$

Example Problem: Job Scheduling



- $p_{mj} = \mathit{True}$ if job j is scheduled on machine m:
  e.g. $p_{11} = p_{14} = p_{23} = p_{22} = \mathit{True}$
- job i starts at t_i and lasts τ_i
- a machine cannot process two or more jobs simultaneously:
  $(p_{mi} \wedge p_{mj}) \rightarrow ((t_i + \tau_i \leq t_j) \vee (t_j + \tau_j \leq t_i)) \quad \Leftrightarrow$
  $(p_{mi} \wedge p_{mj}) \rightarrow ((t_i - t_j \leq -\tau_i) \vee (t_j - t_i \leq -\tau_j))$
- the overall processing time should not exceed T:
  $t_i + \tau_i \leq T \quad \Leftrightarrow \quad t_i - 0 \leq T - \tau_i$

Example Problem: Job Scheduling

$$\phi = \bigwedge_{j=1}^{4} (p_{1j} \vee p_{2j}) \quad \wedge$$

Each task is executed on at least one machine

Example Problem: Job Scheduling

$$\phi = \bigwedge_{j=1}^{4} (p_{1j} \vee p_{2j}) \quad \wedge$$

Each task is executed on at least one machine

$$\bigwedge_{j=1}^{4} ((p_{1j} \to \neg p_{2j}) \wedge (p_{2j} \to \neg p_{1j})) \quad \wedge$$

Each task can be scheduled on one machine only

Example Problem: Job Scheduling

$$\phi = \bigwedge_{j=1}^{4} (p_{1j} \vee p_{2j}) \quad \wedge$$

Each task is executed on at least one machine

$$\bigwedge_{j=1}^{4} ((p_{1j} \rightarrow \neg p_{2j}) \wedge (p_{2j} \rightarrow \neg p_{1j})) \quad \wedge$$

Each task can be scheduled on one machine only

$$\bigwedge_{j=1}^{4} (t_j \geq 0) \;\wedge\; \bigwedge_{j=1}^{4} (t_j \leq T - \tau_j) \quad \wedge$$

General time constraints

Example Problem: Job Scheduling

$$\phi = \bigwedge_{j=1}^{4} (p_{1j} \vee p_{2j}) \quad \wedge$$

Each task is executed on at least one machine

$$\bigwedge_{j=1}^{4} ((p_{1j} \rightarrow \neg p_{2j}) \wedge (p_{2j} \rightarrow \neg p_{1j})) \quad \wedge$$

Each task can be scheduled on one machine only

$$\bigwedge_{j=1}^{4} (t_j \geq 0) \;\wedge\; \bigwedge_{j=1}^{4} (t_j \leq T - \tau_j) \quad \wedge$$

General time constraints

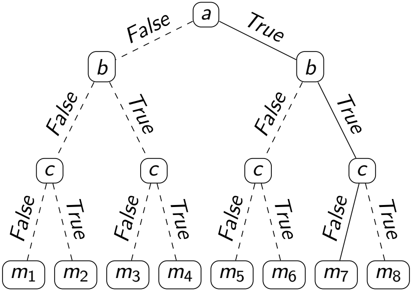
$$\bigwedge_{m=1}^{2} \bigwedge_{i=1}^{3} \bigwedge_{j=i+1}^{4} ((p_{mi} \wedge p_{mj}) \rightarrow ((t_i - t_j \leq -\tau_i) \vee (t_j - t_i \leq -\tau_j)))$$

No time overlap rule

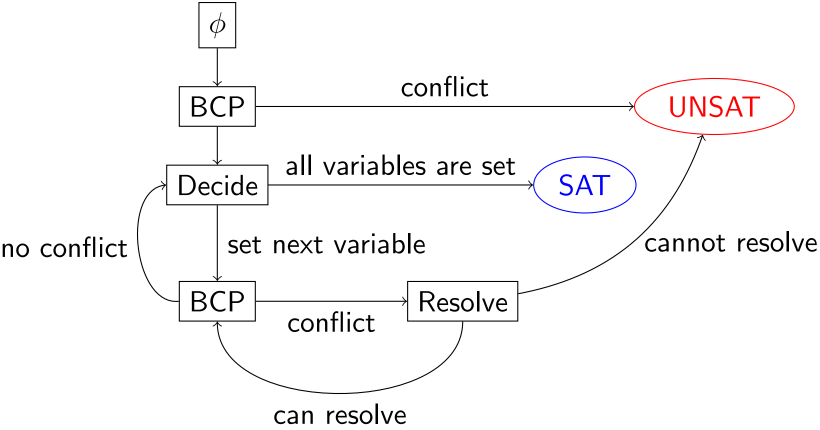
SAT Checking

$\phi = (a \vee b) \wedge (b \vee c) \wedge (c \vee a) \wedge \ldots$

SAT checking = intelligent search in the model space. The model space can be represented as a tree.



SAT Checking



SAT Checking

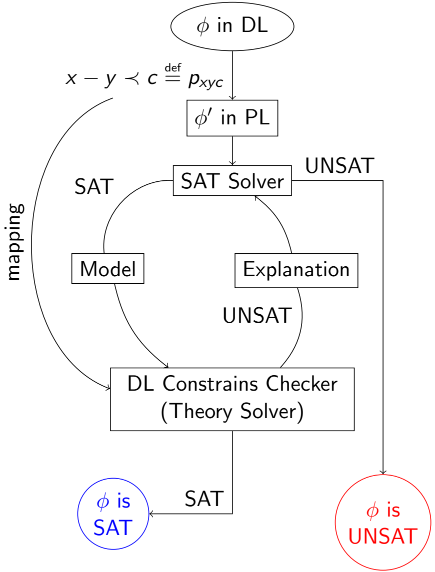


Figure: Lazy approach

SAT Checking

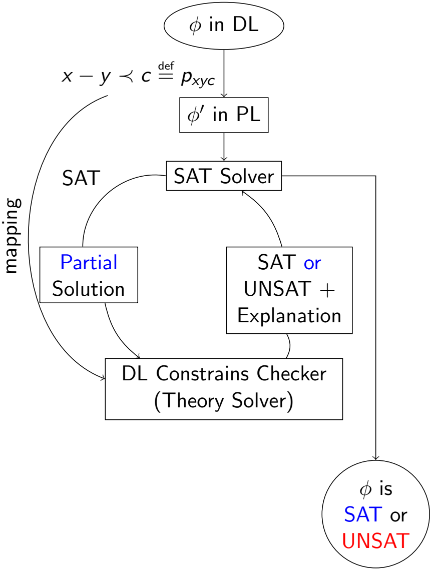
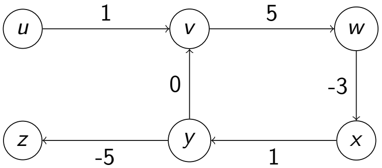


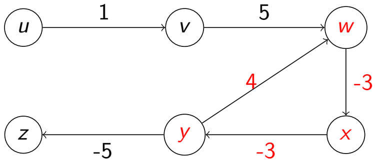
Figure: Incremental approach

Constraint Graph And Negative Cycles



$$(u - v < 1)$$
$$\wedge\ (v - w < 5)$$
$$\wedge\ (w - x \leq -3)$$
$$\wedge\ (x - y < 1)$$
$$\wedge\ (y - z \leq -5)$$
$$\wedge\ (y - v \leq 0)$$

SAT



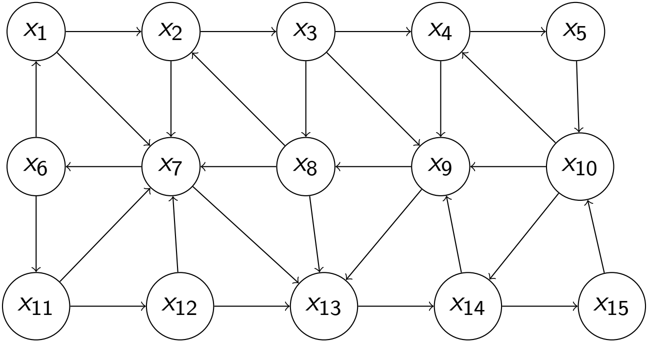
$$(u - v < 1)$$
$$\wedge\ (v - w < 5)$$
$$\wedge\ (w - x \leq -3)$$
$$\wedge\ (x - y < -3)$$
$$\wedge\ (y - z \leq -5)$$
$$\wedge\ (y - w < 4)$$

UNSAT : $0 < -2$

Constraint Graph And Negative Cycles

- First idea: enumerate all cycles
  - and check if they are negative
    i.e. correspond to conflicts like e.g. $0 < -1$, $0 \leq -5$ etc.
  - or they have zero weight and an edge with a strict inequality
    i.e. correspond to $0 < 0$ conflict.
- Any problems with this approach?

Constraint Graph And Negative Cycles



- Problem: a graph can have an enormous number of cycles
- E.g. extreme case: fully connected *directed* graph with n vertices
  - Number of simple cycles $= \sum_{i=2}^{n} \binom{i}{n} \cdot (i-1)!$
  - Factorial grows even faster than exponent $\Rightarrow$ the problem becomes intractable.

Constraint Graph And Negative Cycles

- Use Bellman-Ford algorithm for this task [Cormen et al. 2009]
  - $O(|V| \cdot |E|)$ time complexity
  - $s \in V$ – source vertex (can be selected e.g. randomly)
  - $\Gamma = (V, E, weight)$ – directed graph
  - $d \in V \mapsto \mathbb{R}$ – distance estimate function
  - Note: the algorithm runs once for the whole graph, not once for each vertex (i.e. $|V|$ times).

$\text{Bellman-Ford}(\Gamma, s)$

```
set  d(x) = ∞   ∀x ∈ V
d(s) = 0
```

Constraint Graph And Negative Cycles

- Use Bellman-Ford algorithm for this task [Cormen et al. 2009]
  - $O(|V| \cdot |E|)$ time complexity
  - $s \in V$ – source vertex (can be selected e.g. randomly)
  - $\Gamma = (V, E, weight)$ – directed graph
  - $d \in V \mapsto \mathbb{R}$ – distance estimate function
  - Note: the algorithm runs once for the whole graph, not once for each vertex (i.e. $|V|$ times).

$\textsc{Bellman-Ford}(\Gamma, s)$

set $d(x) = \infty \quad \forall x \in V$
$d(s) = 0$
for $i = 1$ to $|V| - 1$
do for each edge $(x, y) \in E$
    do if $d(x) + weight(x, y) < d(y)$
        then $d(y) = d(x) + weight(x, y)$

Constraint Graph And Negative Cycles

- Use Bellman-Ford algorithm for this task [Cormen et al. 2009]
  - $O(|V| \cdot |E|)$ time complexity
  - $s \in V$ – source vertex (can be selected e.g. randomly)
  - $\Gamma = (V, E, weight)$ – directed graph
  - $d \in V \mapsto \mathbb{R}$ – distance estimate function
  - Note: the algorithm runs once for the whole graph, not once for each vertex (i.e. $|V|$ times).

$\textsc{Bellman-Ford}(\Gamma, s)$

set $d(x) = \infty \quad \forall x \in V$
$d(s) = 0$
for $i = 1$ to $|V| - 1$
do for each edge $(x, y) \in E$
    do if $d(x) + weight(x, y) < d(y)$
        then $d(y) = d(x) + weight(x, y)$
for each edge $(x, y) \in E$
do if $d(x) + weight(x, y) < d(y)$
      then return *False* ($\Rightarrow$ negative cycle detected)

Constraint Graph And Negative Cycles

- Use Bellman-Ford algorithm for this task [Cormen et al. 2009]
  - $O(|V| \cdot |E|)$ time complexity
  - $s \in V$ – source vertex (can be selected e.g. randomly)
  - $\Gamma = (V, E, weight)$ – directed graph
  - $d \in V \mapsto \mathbb{R}$ – distance estimate function
  - Note: the algorithm runs once for the whole graph, not once for each vertex (i.e. $|V|$ times).

$\textsc{Bellman-Ford}(\Gamma, s)$

set $d(x) = \infty \quad \forall x \in V$
$d(s) = 0$
for $i = 1$ to $|V| - 1$
do for each edge $(x, y) \in E$
    do if $d(x) + weight(x, y) < d(y)$
        then $d(y) = d(x) + weight(x, y)$
for each edge $(x, y) \in E$
do if $d(x) + weight(x, y) < d(y)$
    then return *False* ($\Rightarrow$ negative cycle detected)
return *True* ($\Rightarrow$ no negative cycles)

Constraint Graph And Negative Cycles

- [Cotton et al. 2004] uses the admissible graph Γ_d to find a negative or zero weight cycle in the original constraint graph Γ
- Terminology:
  - Reduced cost function $r(x, y) = weight(x, y) + d(x) - d(y)$
  - Admissible edge: $r(x, y) \leq 0$
  - Admissible graph Γ_d – a graph consisting of admissible edges
- Implications:
  - Γ_d is dynamic because it depends on d which changes during the execution of the algorithm
  - If $r(x, y) < 0$ then the edge (x, y) can be "relaxed" i.e. used to improve $d(y)$
  - Γ_d consists of edges which might potentially be used to improve d
  - Intuition: if Γ_d has a cycle then this cycle might be used to update d infinitely

Constraint Graph And Negative Cycles

Theorem

Given a constraint graph Γ
and
a series of distance estimating functions $(d_0, d_1, d_2, d_3, \dots)$*,*
Γ *has a negative or zero cycle*
if and only if
Γ_d *has a cycle under some distance estimate* d_k*.*

Proof.

see [Cotton et al. 2004]. □

Note: the series $(d_0, d_1, d_2, d_3, \dots)$
correspond to the updates of d during the run of Bellman-Ford
where $d_0(s) = 0$ and $d_0(x) = \infty\ \forall x \in V\ s.t.\ x \neq s$
are the initial distance estimates.

Constraint Graph And Negative Cycles

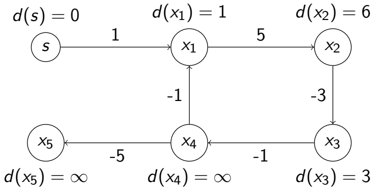


Figure: Γ

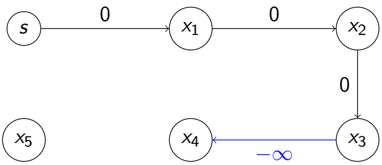


Figure: Γ_d

$$r(x, y) = weight(x, y) + d(x) - d(y)$$

$$\begin{array}{ccc} r(s, x_1) = 0 & r(x_1, x_2) = 0 & r(x_2, x_3) = 0 \\ r(x_3, x_4) = -\infty & r(x_4, x_1) = \infty & r(x_4, x_5) = \varnothing \end{array}$$

Γ_d has no cycles. Let us relax the edge (x_3, x_4)

Constraint Graph And Negative Cycles

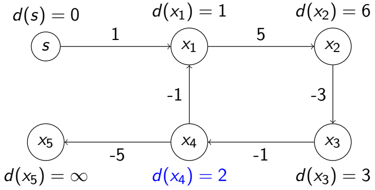


Figure: Γ

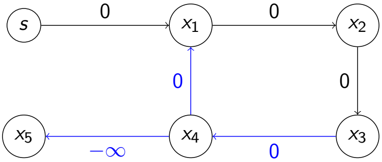


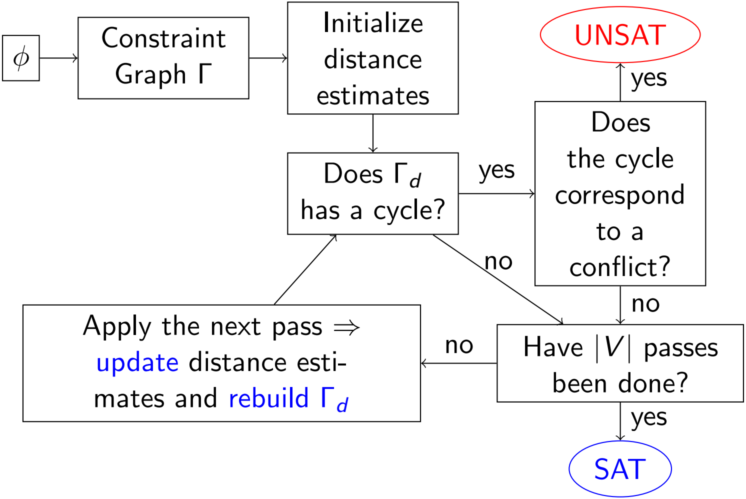
Figure: Γ_d

$$r(x, y) = weight(x, y) + d(x) - d(y)$$

$$\begin{array}{ccc} r(s, x_1) = 0 & r(x_1, x_2) = 0 & r(x_2, x_3) = 0 \\ r(x_3, x_4) = 0 & r(x_4, x_1) = 0 & r(x_4, x_5) = -\infty \end{array}$$

Now Γ_d has a cycle: $x_1 \to x_2 \to x_3 \to x_4 \to x_1$. This cycle in Γ has indeed zero weight.

Constraint Graph And Negative Cycles



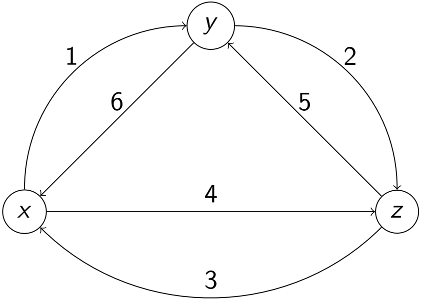
Conclusion

- Many timing problems (logistics, planning, scheduling, circuits checking) can be expressed in DL. Therefore it is important to have an efficient algorithm for checking SAT of a DL formula.
- Conjunction of DL constraints can be represented by a constraint graph Γ.
  - A negative cycle corresponds to a conflict $0 \prec c$ where $c < 0$ and $\prec \in \{<, \leq\}$ e.g. $0 < -3$, $0 \leq -1$ etc.
  - A zero weight cycle with a strict inequality edge corresponds to a conflict $0 < 0$.
- There is no need to enumerate all cycles in Γ. Bellman-Ford algorithm can be used to detect a negative cycle in $O(|V| \cdot |E|)$ operations.
- A cycle in admissible graph Γ_d corresponds to a negative or zero weight cycle in the corresponding constraint graph Γ.

Thank you

Thank you for your attention

Backup Slide. Number of simple cycles formula explained



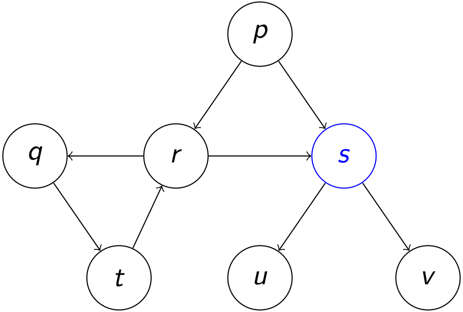
- a fully connected *directed* graph with n vertices.
- Number of *simple* cycles $= \sum_{i=2}^{n} \binom{i}{n} \cdot (i-1)!$

- Example for $i = 3$ (Figure on the left)
- There are $i! = 6$ permutations of the vertices which describe 2 cycles:

$$(x, y, z), (y, z, x), (z, x, y)$$
$$(x, z, y), (z, y, x), (y, x, z)$$

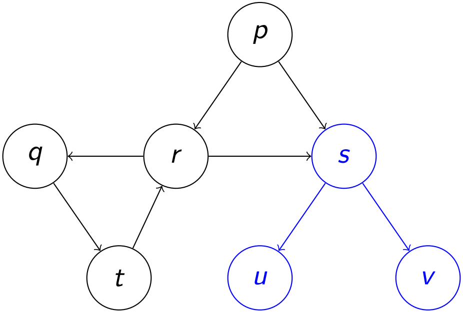
- Each cycle is described by i permutations which can be produced from each other by shifting. Therefore, there are $\frac{i!}{i} = (i-1)!$ cycles.

Backup Slide. Multiple runs of Bellman-Ford



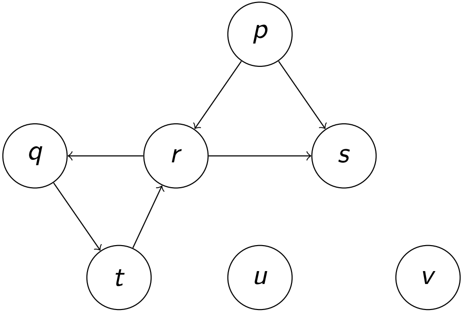
In this example each edge has weight -1.
Suppose that s is selected as the source vertex.

Backup Slide. Multiple runs of Bellman-Ford



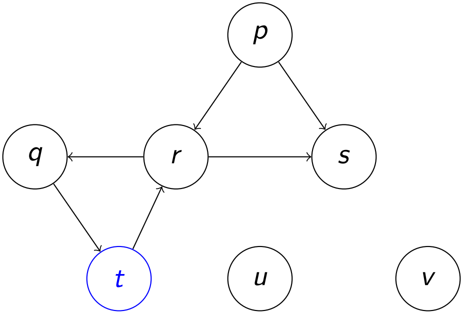
Bellman-Ford processes only a subgraph.
No cycles have been detected.

Backup Slide. Multiple runs of Bellman-Ford



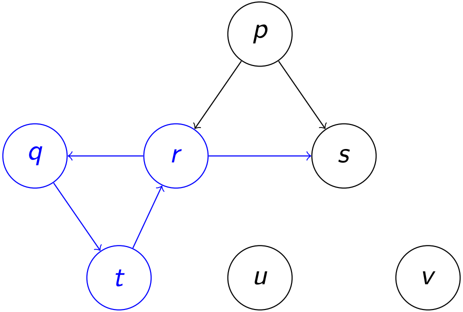
Discard the edges that have been processed
because we do not need to process them twice.
Select the next source vertex and run Bellman-Ford again.

Backup Slide. Multiple runs of Bellman-Ford



Suppose, the next source vertex is t.

Backup Slide. Multiple runs of Bellman-Ford



Bellman-Ford finds the negative cycle $t \to r \to q \to t$.
Since the processed edges have been discarded,
we do not process the edges $(s; u)$ and $(s; v)$ again,
and therefore the complexity for the whole graph stays
$O(|V| \cdot |E|)$
i.e. multiple runs of Bellman-Ford do not increase it.