HELIOS
Execution Optimization For Link Discovery

Axel-Cyrille Ngonga Ngomo

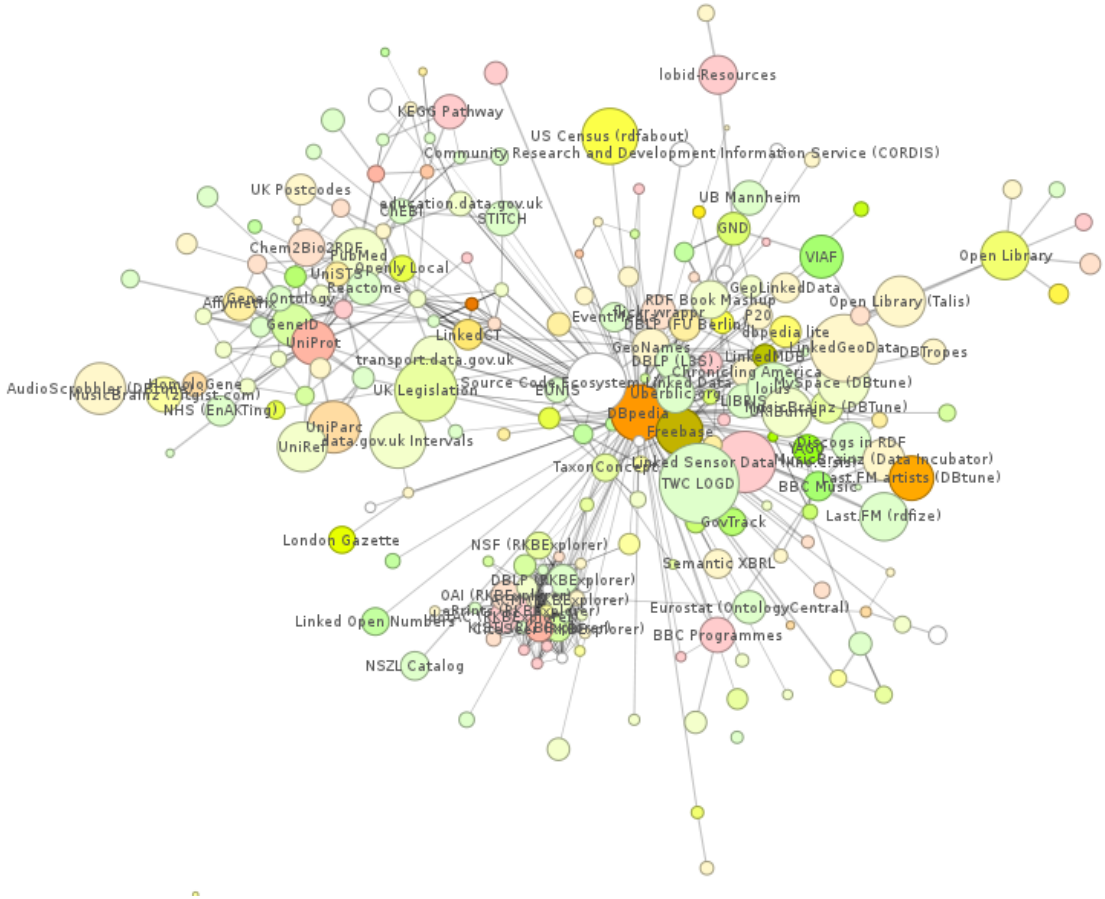
University of Leipzig
Computer Science Department
AKSW Research Group

ISWC 2014
Riva Del Garda, Italy

October 24th, 2014

Why Link Discovery?

1. Fourth principle
2. Links are central for
   - Cross-ontology QA
   - Data Integration
   - Reasoning
   - Federated Queries
   - ...
3. Current topology of the LOD Cloud
   - 62+ billion triples
   - $\approx$ 0.5 billion links
   - Mostly `owl:sameAs`



Formal Definition

Definition (Link Discovery)

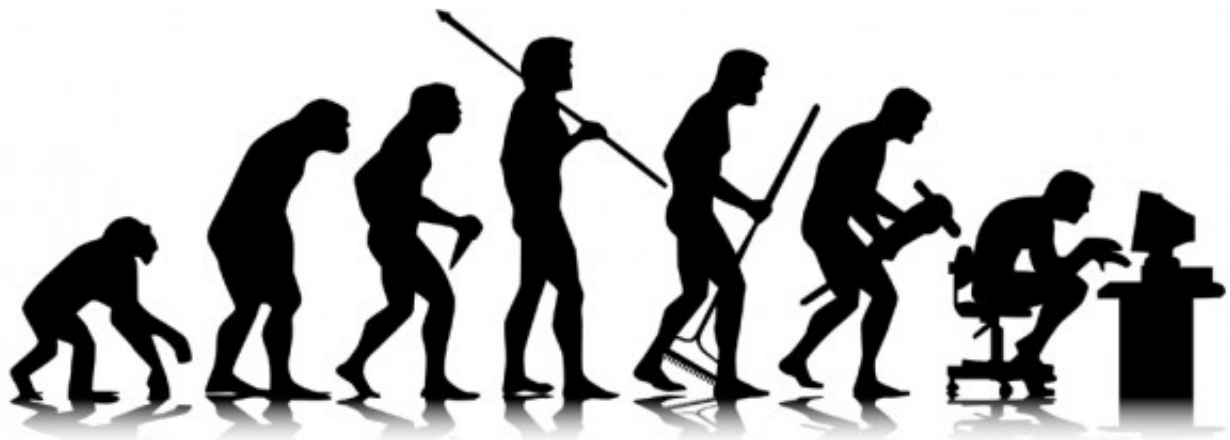
- Given sets S and T of resources and relation $\mathcal{R}$
- Find $M = \{(s, t) \in S \times T : \mathcal{R}(s, t)\}$

Formal Definition

Definition (Link Discovery)

- Given sets S and T of resources and relation $\mathcal{R}$
- Find $M = \{(s,t) \in S \times T : \mathcal{R}(s,t)\}$

- Automatic computation of M most commonly difficult
- Too tedious for manual evaluation on large knowledge bases



Normal Form for Link Specifications

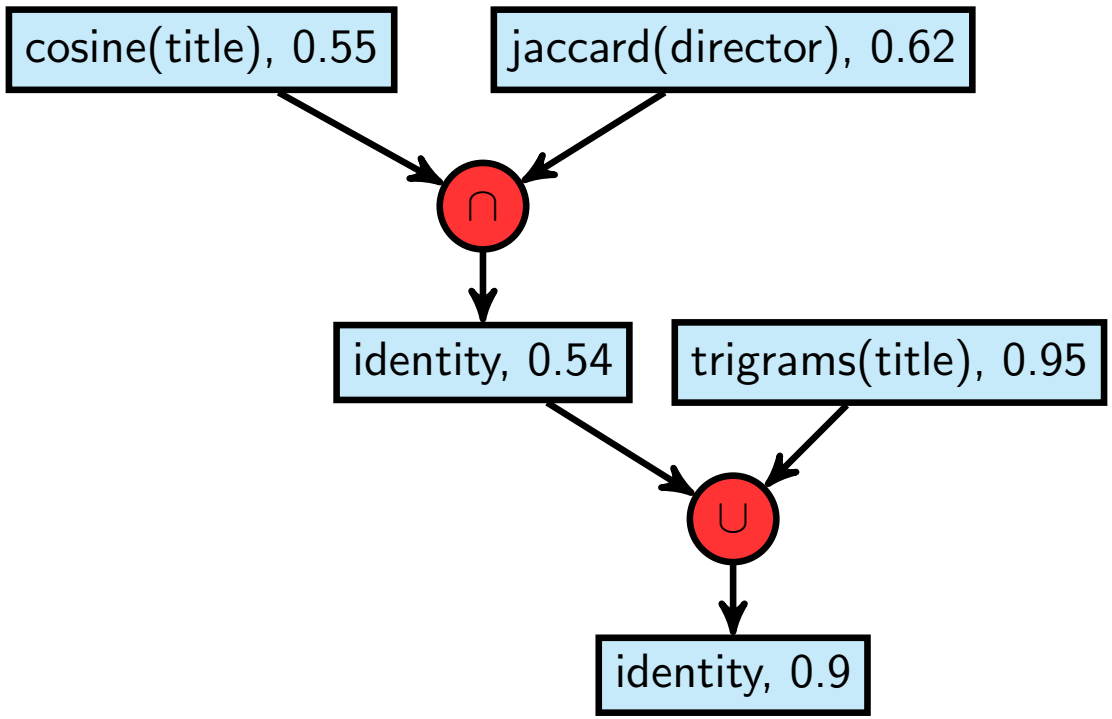
Definition (Link Discovery (Approximation))

- Find $M' = \{(s, t) \in S \times T : \sigma(s, t) \geq \theta\}$

Normal Form for Link Specifications

Definition (Link Discovery (Approximation))

- Find $M' = \{(s, t) \in S \times T : \sigma(s, t) \geq \theta\}$



- σ can be complex and represented as bi-partite tree
- Atomic elements are **filters** (blue) and **operators** (red)

Why is it difficult?

- **Accuracy**
  - Detection of appropriate filters, thresholds and operators
  - Solutions incl.
    - Unsupervised learning (EUCLID, EAGLE, KnoFuss)
    - Active learning (RAVEN, EAGLE, ActiveGenLink)
    - Supervised (EAGLE, EUCLID)

Why is it difficult?

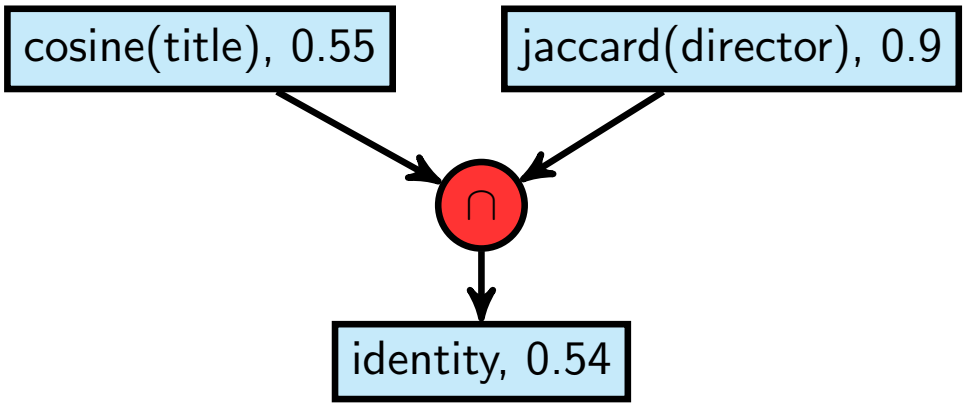
- **Accuracy**
  - Detection of appropriate filters, thresholds and operators
  - Solutions incl.
    - Unsupervised learning (EUCLID, EAGLE, KnoFuss)
    - Active learning (RAVEN, EAGLE, ActiveGenLink)
    - Supervised (EAGLE, EUCLID)

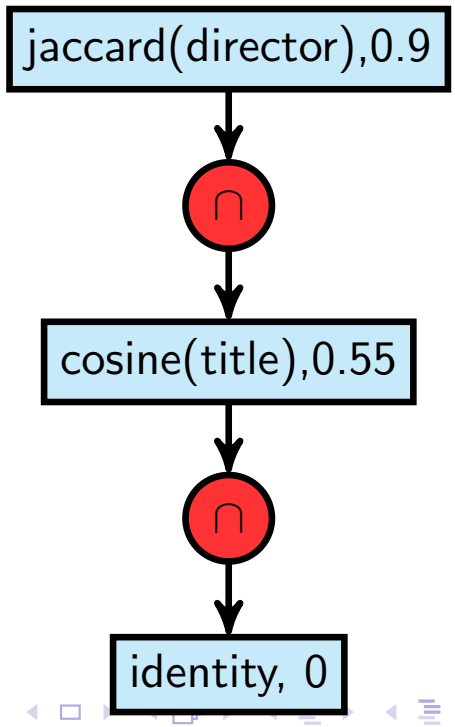
- **Time complexity**
  - Quadratic a-priori runtime
  - So far optimization of runtime for single filters (HR^3, HYPPO, PPJoin+, LIMES)

Execution Optimization

HELIOS

- **Goal**: Optimize runtime for **whole specifications**
- **Approach**:
  1. Rewrite specifications to small equivalent specifications
  2. Optimize execution by finding time-efficient plans

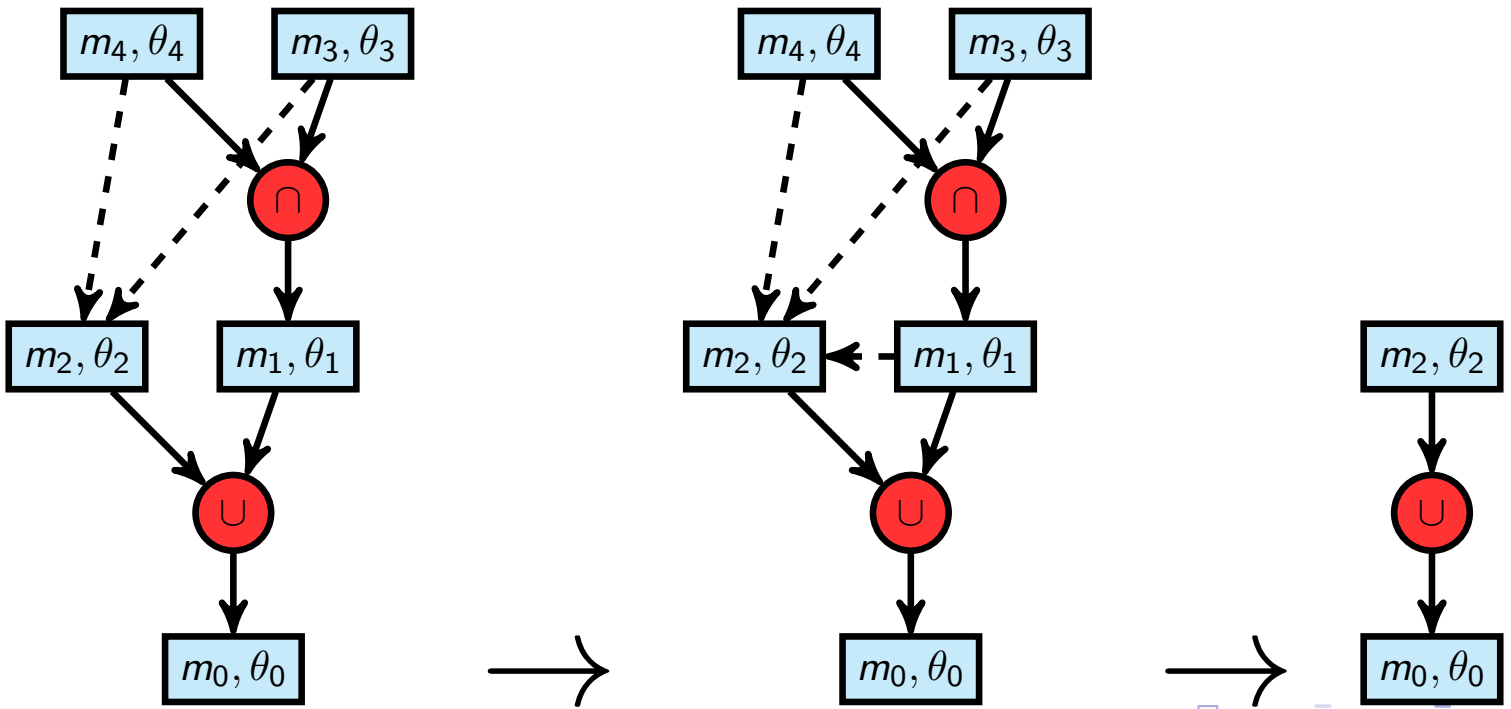




Rewriting

Overview

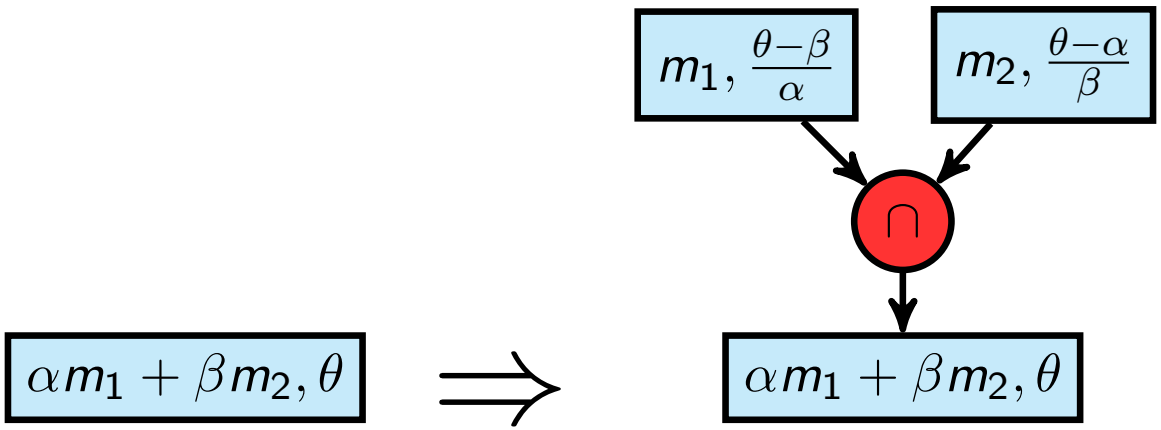
- Fix-point operator
- Three steps
  1. Leaf generation
  2. Dependency detection and propagation
  3. Reduction



Rewriting

Step 1: Leaf generation

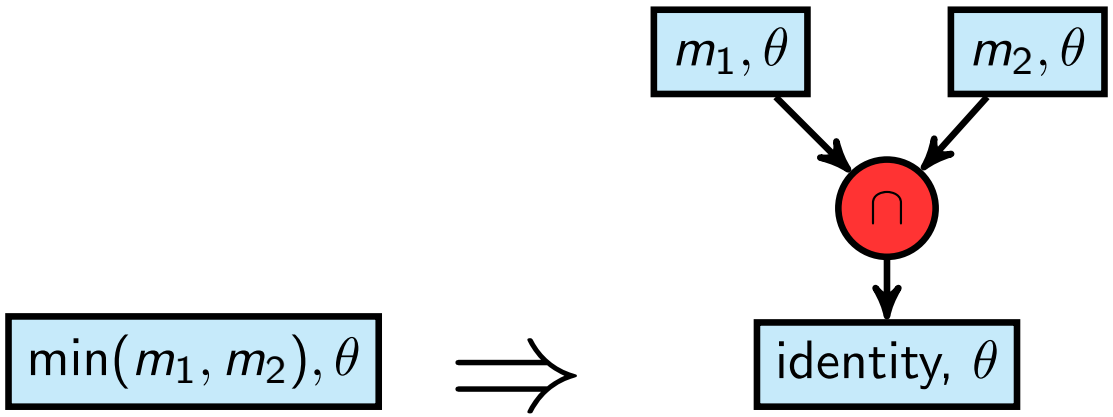
- Use equivalence between operators to generate atomic leaves
- Make use of existence of time-efficient algorithms for atomic measures



Rewriting

Step 1: Leaf generation

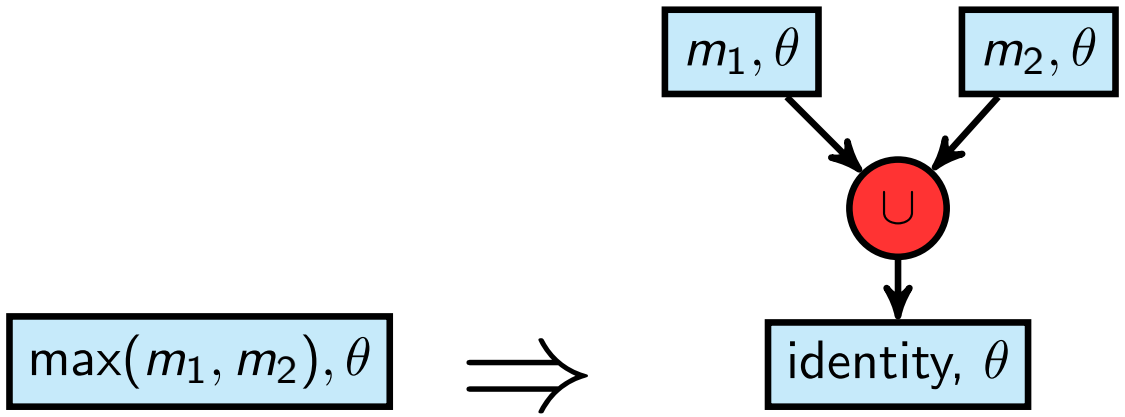
- Use equivalence between operators to generate atomic leaves
- Make use of existence of time-efficient algorithms for atomic measures



Rewriting

Step 1: Leaf generation

- Use equivalence between operators to generate atomic leaves
- Make use of existence of time-efficient algorithms for atomic measures

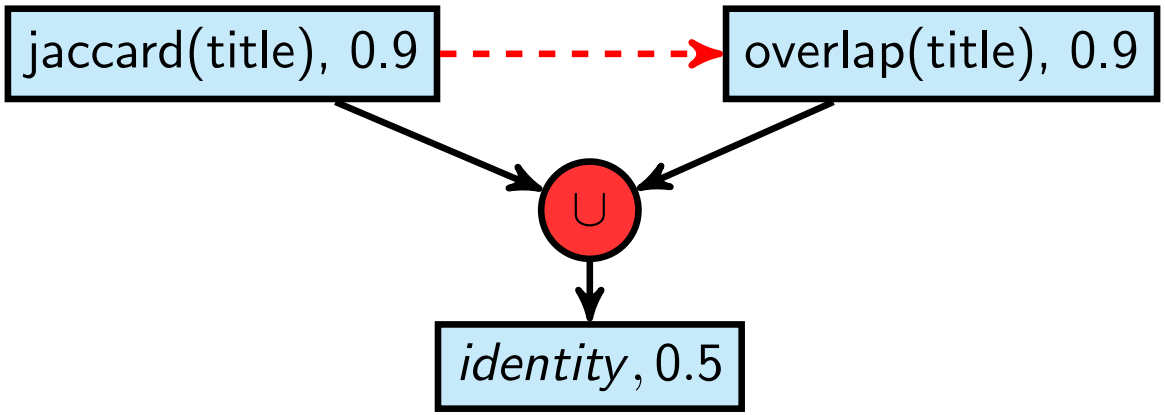


Step 2: **Dependency Detection** and Propagation

- **Idea**: $\mathit{depends}(L_1, L_2) \leftrightarrow \forall S, T : \mathit{result}(L_1) \subseteq \mathit{result}(L_2)$
- Use dependencies between measures, e.g.
  - $\theta_1 \geq \theta_2 \rightarrow \mathit{depends}((m, \theta_1), (m, \theta_2))$
  - $\texttt{jaccard}(x, y) \geq \theta \rightarrow \texttt{overlap}(x, y) \geq \frac{2\theta}{1+\theta}$

Step 2: **Dependency Detection** and Propagation

- **Idea**: $\mathit{depends}(L_1, L_2) \leftrightarrow \forall S, T : \mathit{result}(L_1) \subseteq \mathit{result}(L_2)$
- Use dependencies between measures, e.g.
  - $\theta_1 \geq \theta_2 \rightarrow \mathit{depends}((m, \theta_1), (m, \theta_2))$
  - $\texttt{jaccard}(x, y) \geq \theta \rightarrow \texttt{overlap}(x, y) \geq \frac{2\theta}{1+\theta}$

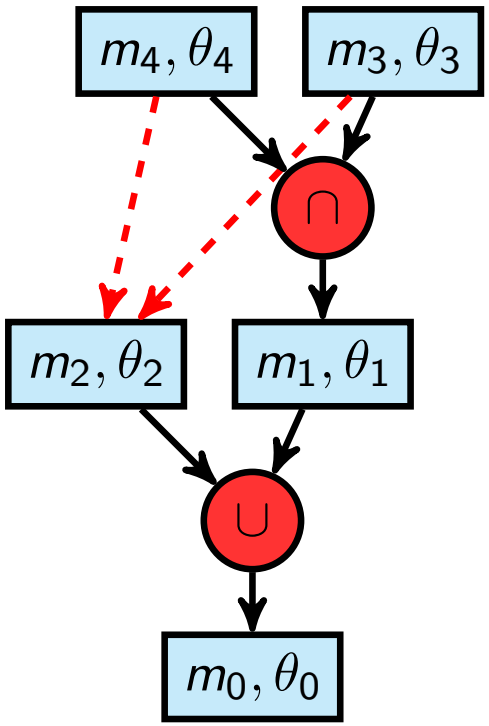


Step 2: **Dependency** Detection and **Propagation**

- **Idea**: Propagate dependencies towards the root
- Use set theory, e.g.,
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cap(L_4, L_3), \theta_1), L_2)$
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cup(L_4, L_3), \theta_1), L_2)$

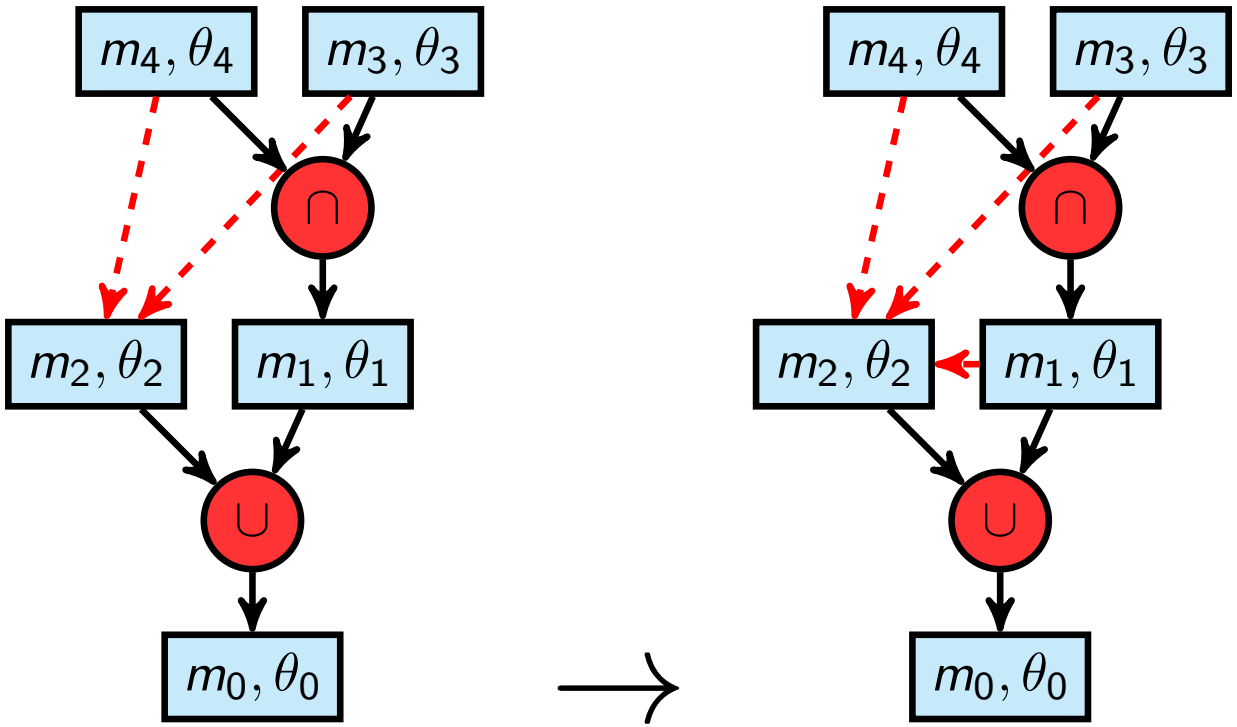
Step 2: **Dependency** Detection and **Propagation**

- **Idea**: Propagate dependencies towards the root
- Use set theory, e.g.,
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cap(L_4, L_3), \theta_1), L_2)$
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cup(L_4, L_3), \theta_1), L_2)$



Step 2: **Dependency** Detection and **Propagation**

- **Idea**: Propagate dependencies towards the root
- Use set theory, e.g.,
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cap(L_4, L_3), \theta_1), L_2)$
  - $depends(L_4, L_2) \wedge depends(L_3, L_2) \rightarrow depends(m_1(\cup(L_4, L_3), \theta_1), L_2)$



Rewriting

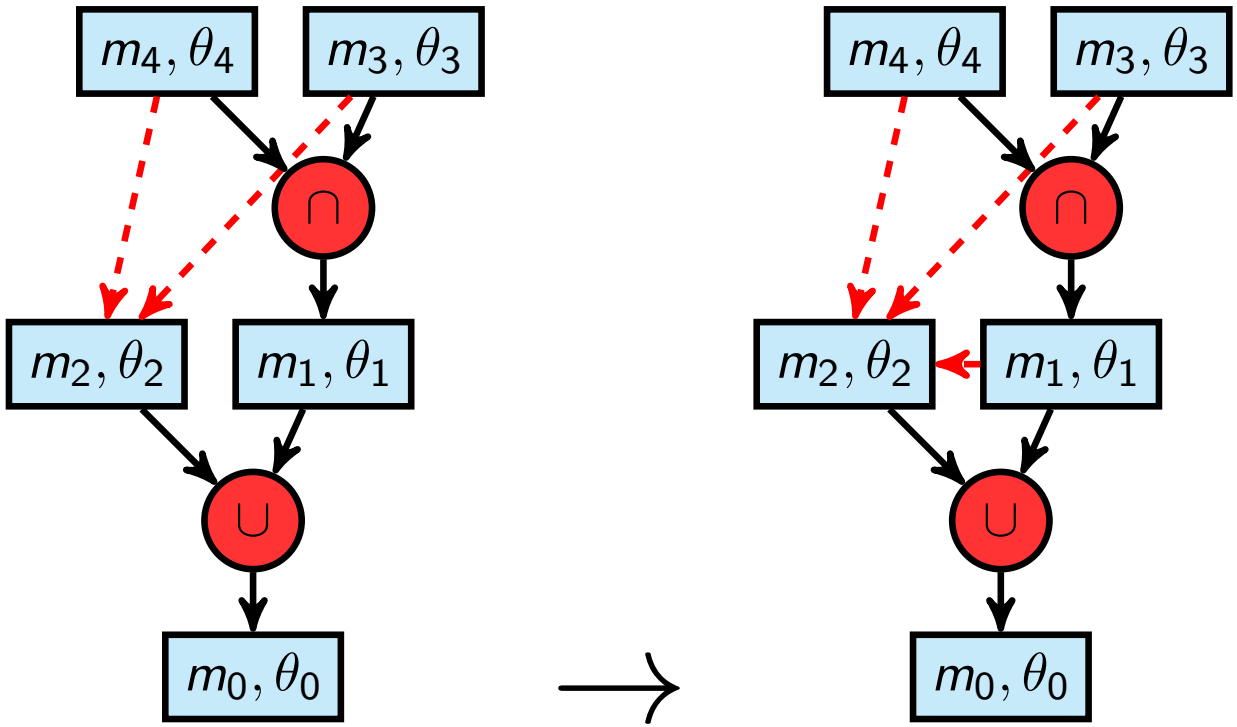
Step 3: Reduction

- **Idea**: Remove unnecessary specs
- Use set theory, e.g.,
  - $depends(L_1, L_2) \wedge L = m(\cap(L_1, L_2), \theta) \Rightarrow L := m(\cap(L_1), \theta)$
  - $depends(L_1, L_2) \wedge L = m(\cup(L_1, L_2), \theta) \Rightarrow L := m(\cup(L_2), \theta)$

Rewriting

Step 3: Reduction

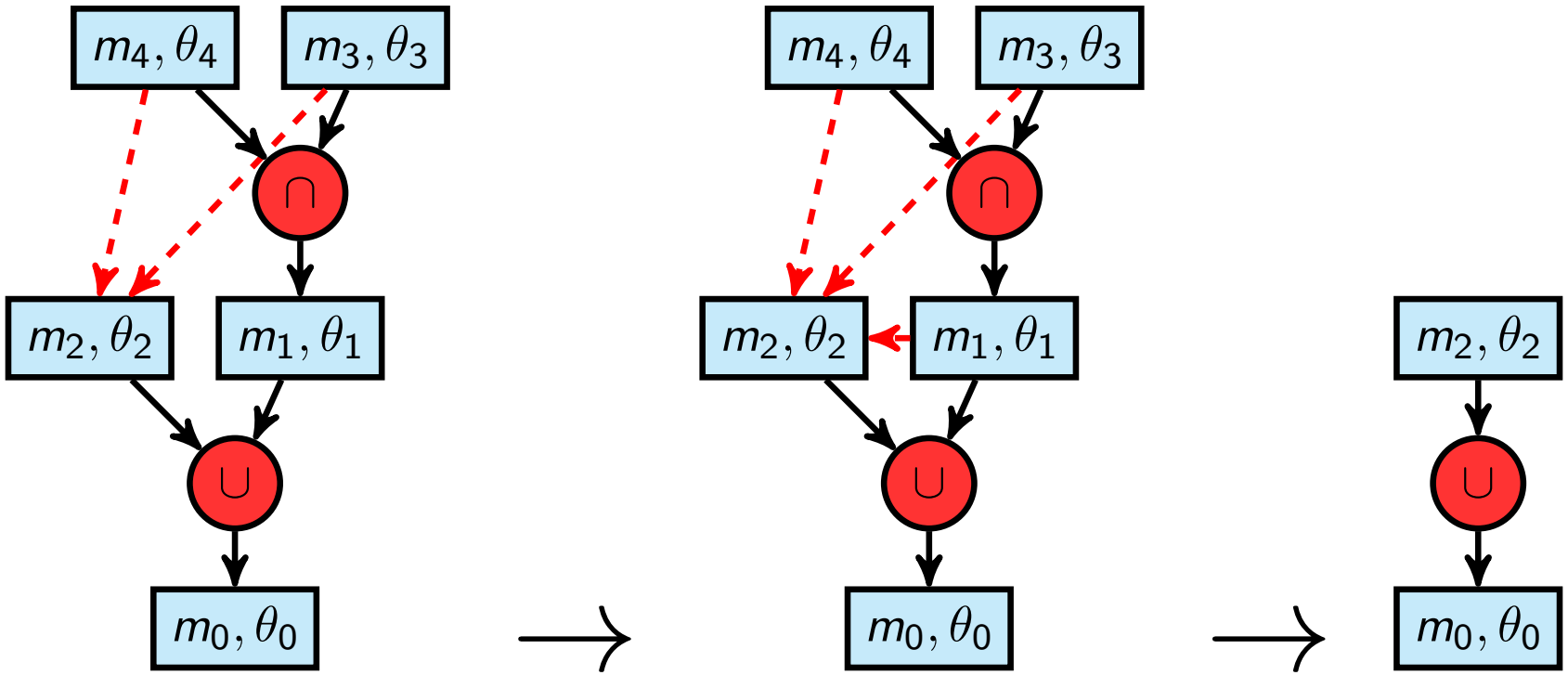
- **Idea**: Remove unnecessary specs
- Use set theory, e.g.,
  - $depends(L_1, L_2) \wedge L = m(\cap(L_1, L_2), \theta) \Rightarrow L := m(\cap(L_1), \theta)$
  - $depends(L_1, L_2) \wedge L = m(\cup(L_1, L_2), \theta) \Rightarrow L := m(\cup(L_2), \theta)$



Rewriting

Step 3: Reduction

- **Idea**: Remove unnecessary specs
- Use set theory, e.g.,
  - $depends(L_1, L_2) \wedge L = m(\cap(L_1, L_2), \theta) \Rightarrow L := m(\cap(L_1), \theta)$
  - $depends(L_1, L_2) \wedge L = m(\cup(L_1, L_2), \theta) \Rightarrow L := m(\cup(L_2), \theta)$



Approach

HELIOS Planner

- **Goal**: Derive equivalent and complete from rewritten specification
- **Required**:
  1. Plan evaluation to detect good plans
  2. Heuristic to reduce number of plans to evaluate
- **Important**:
  1. No a-priori statistics on S and T

Plan Evaluation

Evaluation Functions

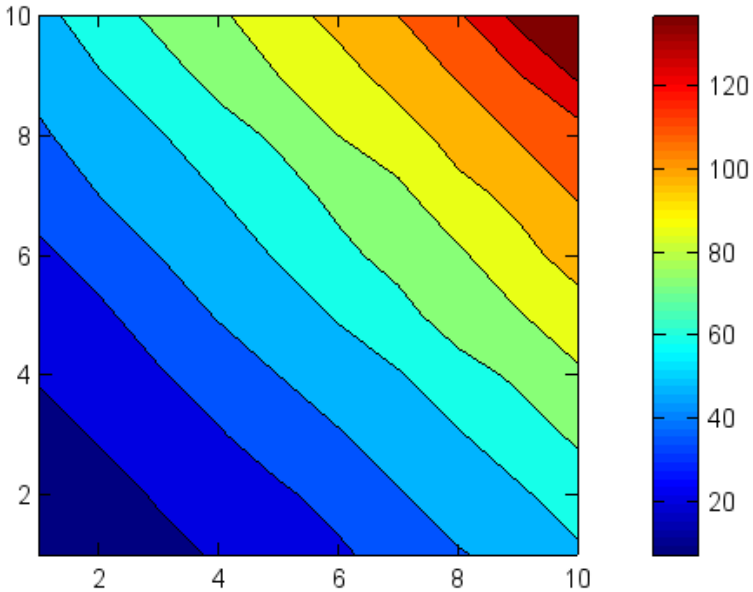
- **Runtime**: How long will the plan need?
- **Selectivity**: How many results will the plan generate?

Plan Evaluation

Evaluation Functions

- **Runtime**: How long will the plan need?
- **Selectivity**: How many results will the plan generate?

- **A-priori Runtime Evaluation**
  1. Measure runtime on samples
  2. Compute perfect numerical solution to linear regression
  3. Derive runtime characteristic for each measure
  4. $\gamma(L) = \sum_{L' \subseteq L} \gamma(L') + \gamma(root(L))$

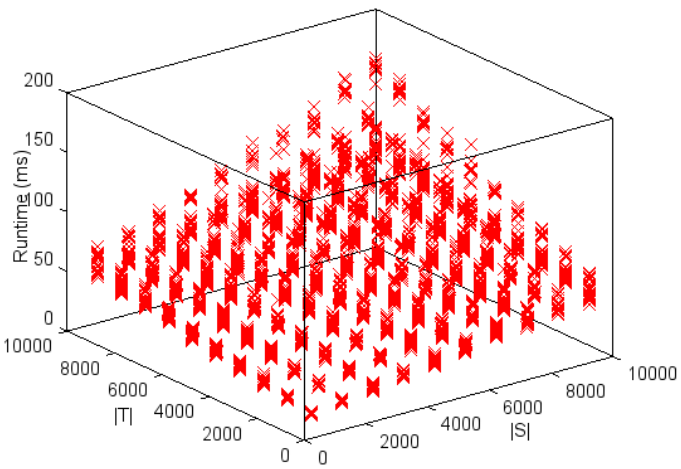


Plan Evaluation

Evaluation Functions

- **Runtime**: How long will the plan need?
- **Selectivity**: How many results will the plan generate?

- **Selectivity**
  - Similar approach
  - Derive probability of pair in input also being in output
  - Assume independence of sub-specifications, e.g.
    - $s(\cap(L_1, L_2)) = s(L_1)s(L_2)$
    - $s(\cup(L_1, L_2)) = 1 - (1 - s(L_1))(1 - s(L_2))$



Plan Optimization

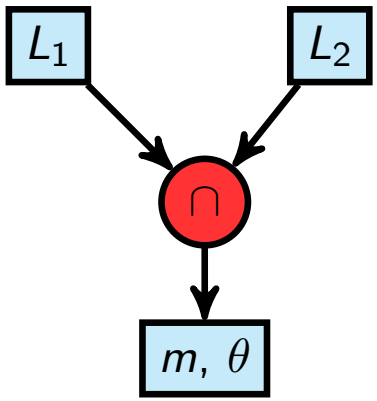
Approach

- If L is atomic, return L
- For $L = m(op(L_1, L_2), \theta)$
  1. Compute plans for L_1 and L_2
  2. Combine plans for L_1 and L_2 to plan which minimizes runtime

Plan Optimization

Approach

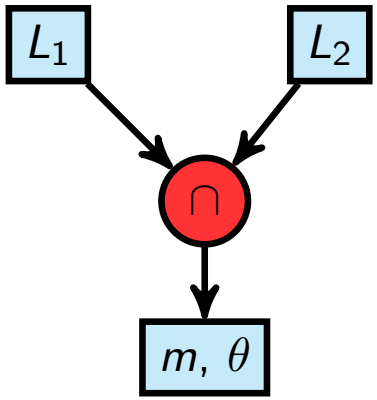
- If L is atomic, return L
- For $L = m(op(L_1, L_2), \theta)$
  1. Compute plans for L_1 and L_2
  2. Combine plans for L_1 and L_2 to plan which minimizes runtime

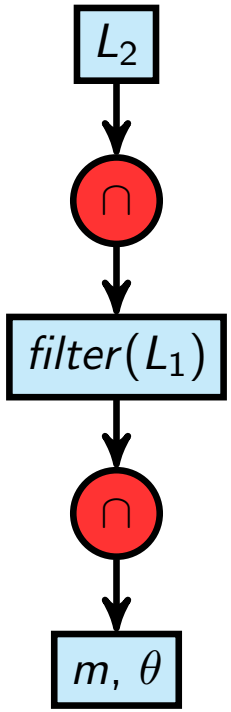


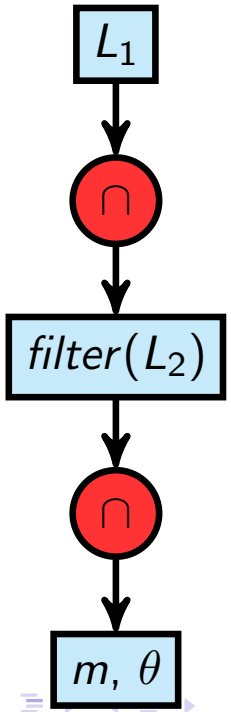
Plan Optimization

Approach

- If L is atomic, return L
- For $L = m(op(L_1, L_2), \theta)$
  1. Compute plans for L_1 and L_2
  2. Combine plans for L_1 and L_2 to plan which minimizes runtime







Experimental Setup

- Implemented in LIMES
- Comparison with canonical LIMES planner
  - Optimized algorithms
  - Time-efficient
- Two types of datasets
  - Manually generated
    - Domain experts
    - 17 specifications
    - 18 datasets
    - Size between 1 and 3
  - Automatically generated
    - EAGLE algorithm
    - 2180 specifications
    - 6 datasets
    - Size between 1 and 11

Runtime on Expert Specifications

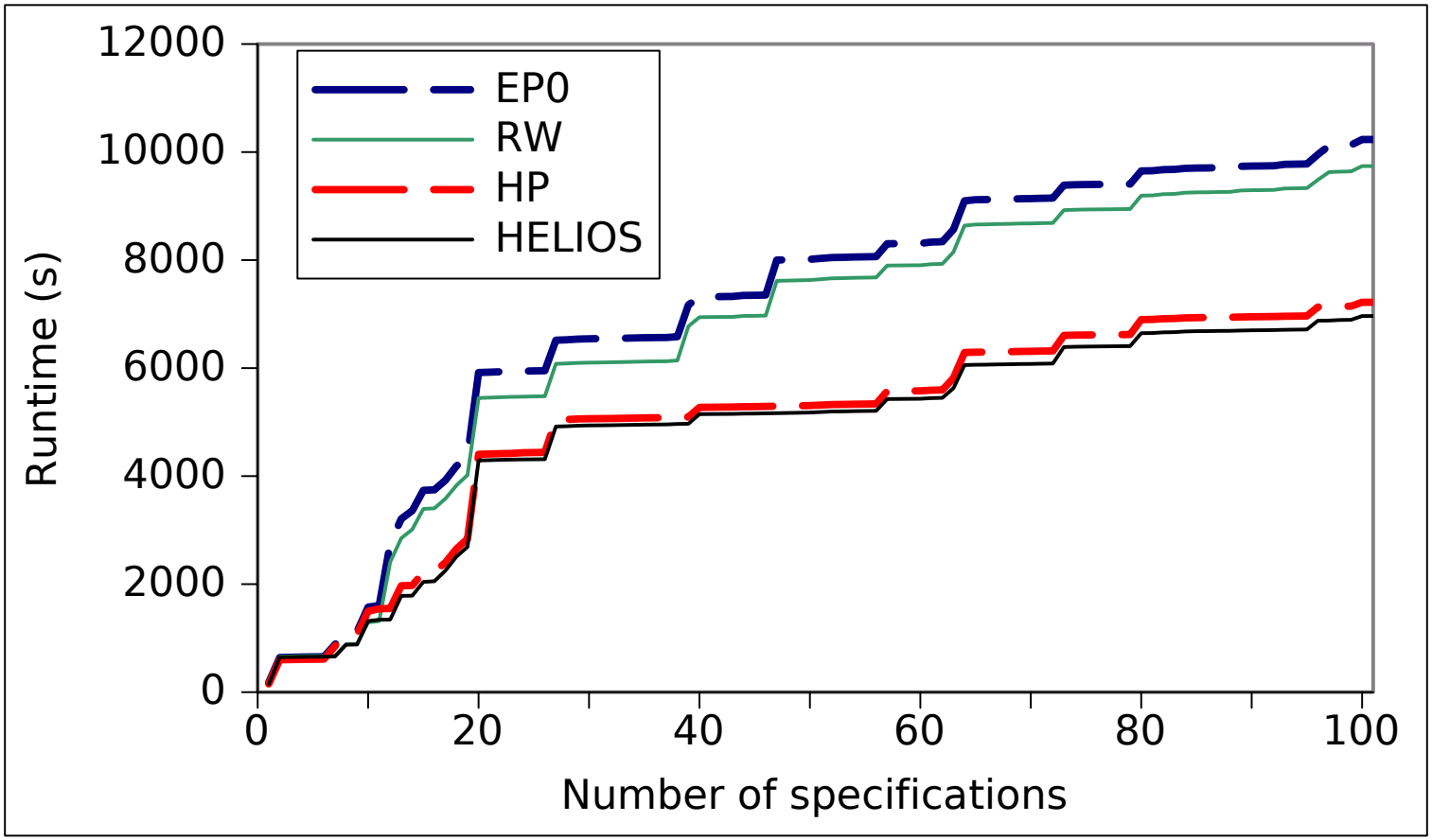
Source - Target	$\|S\| \times \|T\|$	*LIMES* (ms)	*RW* (ms)	*HP* (ms)	HELIOS (ms)	Gain (ms)
Stad - Rmon	341.9×10^3	25	23	15	**14**	11
EVT - DF (E)	531.0×10^3	**893**	906	909	905	-12
Climb - Rail	1.9×10^6	41	**40**	**40**	**40**	1
DBLP - DataSW	92.2×10^6	59	59	58	**54**	5
EVT - DF (P)	148.4×10^6	2,477	2,482	2,503	**2,434**	43
EVT - DBLP	161.0×10^6	9,654	**9,575**	9,613	9,612	42

Runtime on Expert Specifications

Source - Target	$\|S\| \times \|T\|$	*LIMES* (ms)	*RW* (ms)	*HP* (ms)	HELIOS (ms)	Gain (ms)
Stad - Rmon	341.9×10^3	25	23	15	**14**	11
EVT - DF (E)	531.0×10^3	**893**	906	909	905	-12
Climb - Rail	1.9×10^6	41	**40**	**40**	**40**	1
DBLP - DataSW	92.2×10^6	59	59	58	**54**	5
EVT - DF (P)	148.4×10^6	2,477	2,482	2,503	**2,434**	43
EVT - DBLP	161.0×10^6	9,654	**9,575**	9,613	9,612	42
Climb - DBP	312.4×10^3	**55**	**55**	**55**	**55**	0
DBP - LGD (E)	34.1×10^6	2,259	2,133	**1,206**	1,209	1,050
Climb - LGD	215.0×10^6	24,249	24,835	**3,497**	3,521	20,728
DBP - LGD (A)	383.8×10^6	52,663	59,635	1,066	**1,064**	51,599
LGD - LGD	509.3×10^9	46,604	38,560	32,831	**22,497**	24,107

- Runtime on specifications of size 1 not worse
- Significantly better (up to 49.5 times) on specifications of size 3
- HELIOS better than RW or HP alone in most cases

Runtime on LGD



- Significantly outperform the canonical planner
- Effect of rewriting can be clearly seen
- Planner leads to highest gain

Conclusion and Future Work

- Presented HELIOS
- Rule-based rewriting
- Divide-and-conquer approach to planning
- Not yet explored
  - Search-based approaches
  - Backtracking
  - Dynamic planning

That's all Folks!

Thank you!
Questions?

Axel Ngonga
University of Leipzig
AKSW Research Group
Augustusplatz 10, Room P616
04109 Leipzig, Germany
ngonga@informatik.uni-leipzig.de
http://limes.sf.net