SUBQUERY PLAN REUSE BASED QUERY OPTIMIZATION

COMAD 2011

Meduri Venkata Vamsikrishna, Prof. Tan Kian Lee

- Introduction
- Problem Statement
- Related Work
- Sub query plan Reuse (SR) based approach
- Performance Study
- Conclusion
- Future Work
- References

Query optimization

- Dynamic Programming
  - Optimal / best quality plans
  - Exponential search space
- Randomized algorithms
  - Reduced search space
  - Sub optimal plans (Close to optimal if run for a long time)
- Greedy algorithms
  - Minimum search space
  - Sub Optimal plans

Complex queries

- Complex queries typically have large number of tables and clauses (predicates)
  - Query Q5 in TPC-H

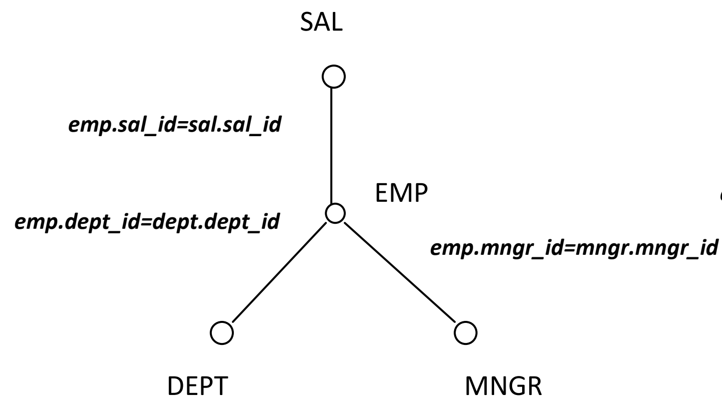
```
select n_name, sum(l_extendedprice * (1 - l_discount)) as
revenue from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey and l_orderkey = o_orderkey and
l_suppkey = s_suppkey and
c_nationkey = s_nationkey and s_nationkey = n_nationkey
and n_regionkey = r_regionkey and
r_name = '[REGION]' and o_orderdate >= date '[DATE]' and
o_orderdate < date '[DATE]' + interval '1' year group by
n_name order by revenue desc;
```

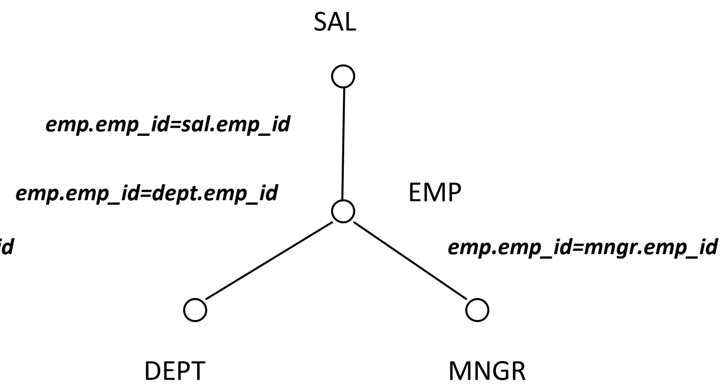
Dense queries: Inferred Predicates

- OLAP queries are dense, TPC-H benchmark is known to contain OLAP queries
- Multi-referenced columns can lead to inferred predicates in PostgreSQL
- Even for star queries, density can increase due to inferred predicates

Query	DP Lattice:
Q1: SELECT COUNT(*) FROM emp, sal, dept, mngr WHERE emp.sal_id = sal.sal_id AND emp.dept_id = dept.dept_id AND emp.mngr_id = mngr.mngr_id;	LEVEL 2: NUM OF QUERY PLANS = 3 LEVEL 3: NUM OF QUERY PLANS = 6 LEVEL 4: NUM OF QUERY PLANS = 3
Q2: SELECT COUNT(*) FROM emp, sal, dept, mngr WHERE emp.emp_id = sal.emp_id AND emp.emp_id = dept.emp_id AND emp.emp_id = mngr.emp_id;	LEVEL 2: NUM OF QUERY PLANS = 6 LEVEL 3: NUM OF QUERY PLANS = 12 LEVEL 4: NUM OF QUERY PLANS = 7

Dense queries: Inferred Predicates (cont'd)

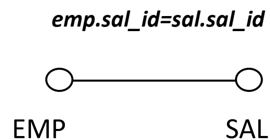


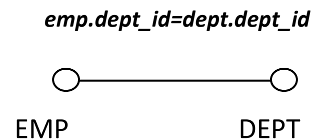


STAR QUERY Q1

STAR QUERY Q2

PLANS FOR "Q1" AT LEVEL 2

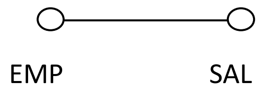




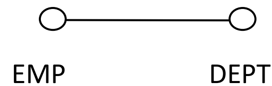
Dense queries: Inferred Predicates

PLANS FOR "Q2" AT LEVEL 2

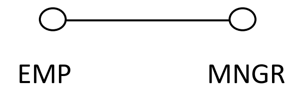
emp.emp_id=sal.emp_id



emp.emp_id=dept.emp_id

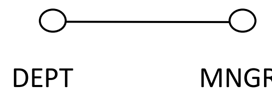


emp.emp_id=mngr.emp_id

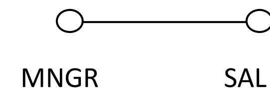


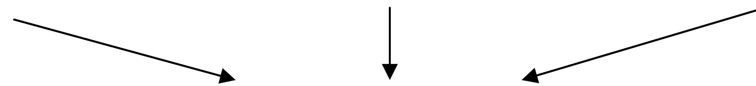
sal.emp_id=dept.emp_id

dept.emp_id=mngr.emp_id

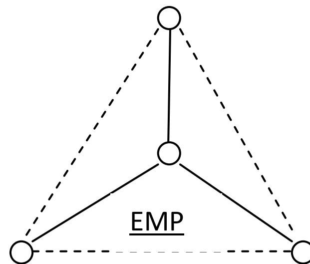


mngr.emp_id=sal.emp_id





SAL



DEPT MNGR

Inferred predicates in "Q2"

Optimality Vs Scalability

- Dynamic Programming (DP) runs out of memory
  - search space enumeration of DP is infeasible as the number of tables and clauses in the query go higher

- $\forall$ i=0 to n−1, $|Plansi| \geq nCi$

- It is essential to strike a balance between scalability and optimality

Problem Statement

- Our scheme is aimed at generating the search space efficiently and to bring about

  an optimal plan.

- Here is our problem statement

  - "Optimization of complex queries to obtain high quality plans using Dynamic

  Programming based algorithms"

- Our Approach:

  - Generate a part of the search space and reuse it for the remaining fraction

  - Detection of similar sub queries and plan construction by reuse

Dynamic Programming

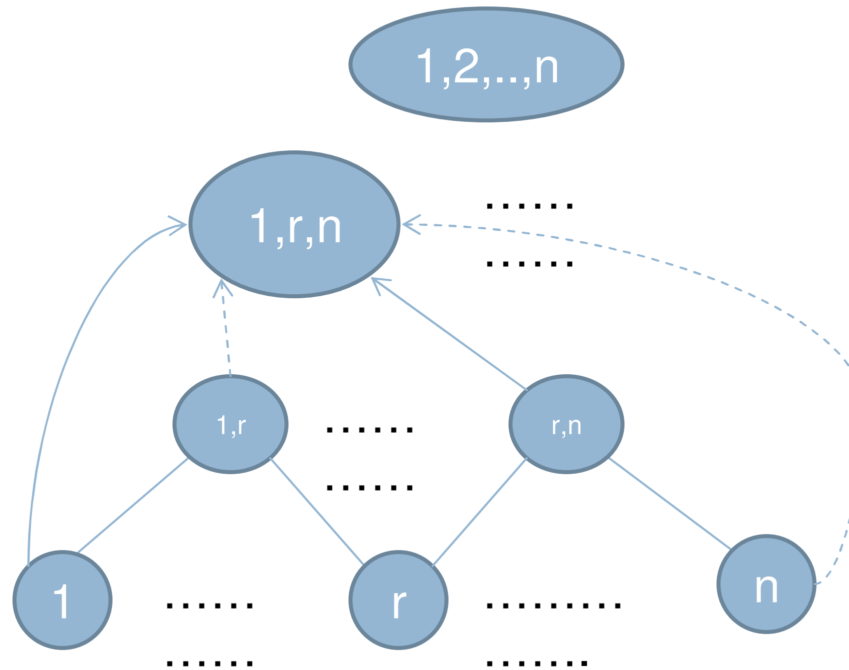
- Dynamic Programming uses a lattice of “n” levels where “n” is the number of relations in the query (with cartesian product avoidance, |plans| can be lesser)

level “n”
nCn=1

level 3 < = nC3

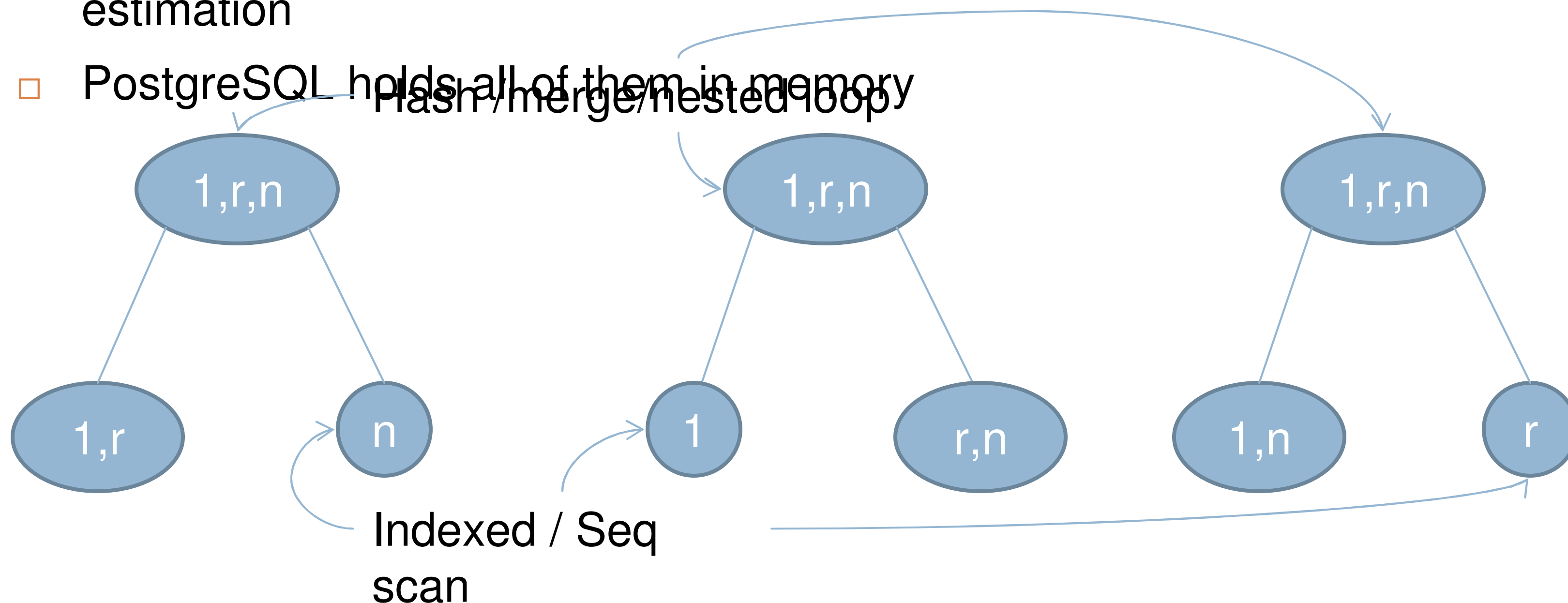
level 2 < =nC2

level 1
nC1



Various plans for a join candidate

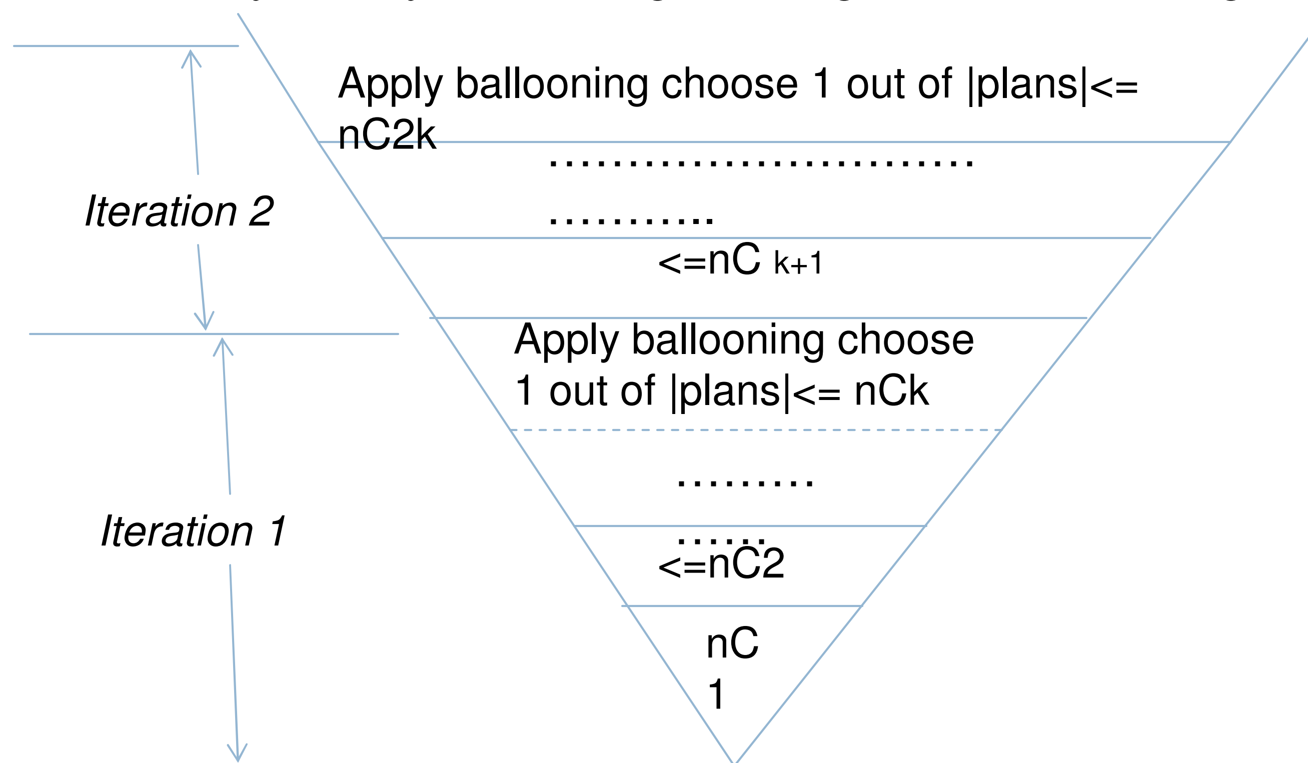
- Keep a record of all the sub plans
- Various join methods, many join orders , choose best based on cost estimation
- PostgreSQL holds all of them in memory



Total plans = $nC_1 + nC_2 + nC_3$ * (number of plans for various join orders) + ………… + $nC_n > 2^n$

Iterative Dynamic Programming (IDP)

- Iteratively run Dynamic Programming interleaved with greedy pruning



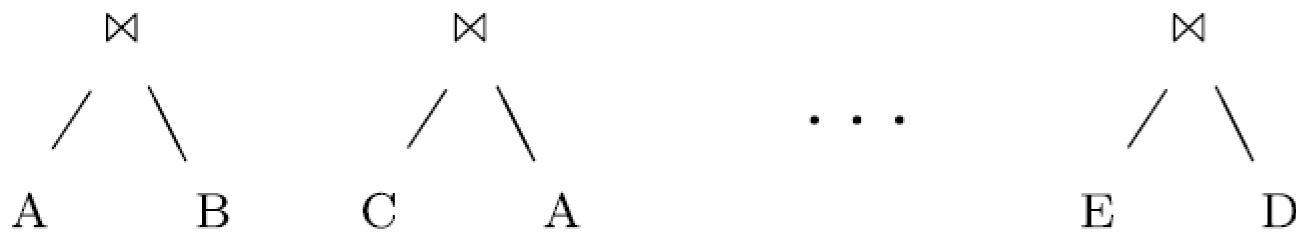
Significance of “k”

- Value of “k” decides optimality
  - Since the number of pure DP rounds are regulated by k
- For most cases (unless the query is exorbitant that DP cannot finish even one level in the lattice), there exists a suitable value of “k” for which IDP runs to completion
- The higher the “K”, better is the plan quality
  - Each DP round gets longer, less number of DP rounds, less number of greedy interventions
- Ballooning – Looking ahead for best plans
  - Greedily increase the subplans and compare the individually obtained complete plans on cost metric, delete all except the best and their subplans too
- Our scheme is aimed at increasing the value of “k”

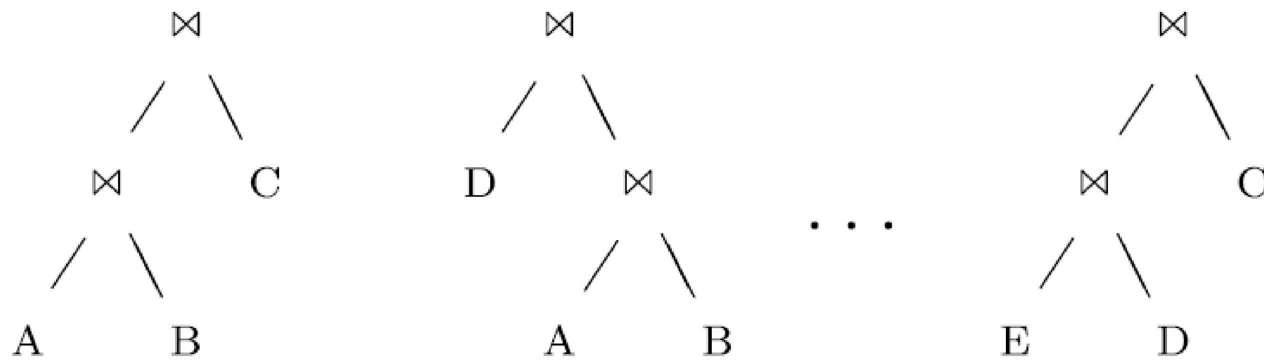
IDP: Optimizing an example query

A B C D E

Step 1: access plans



Step 2: 2-way joins

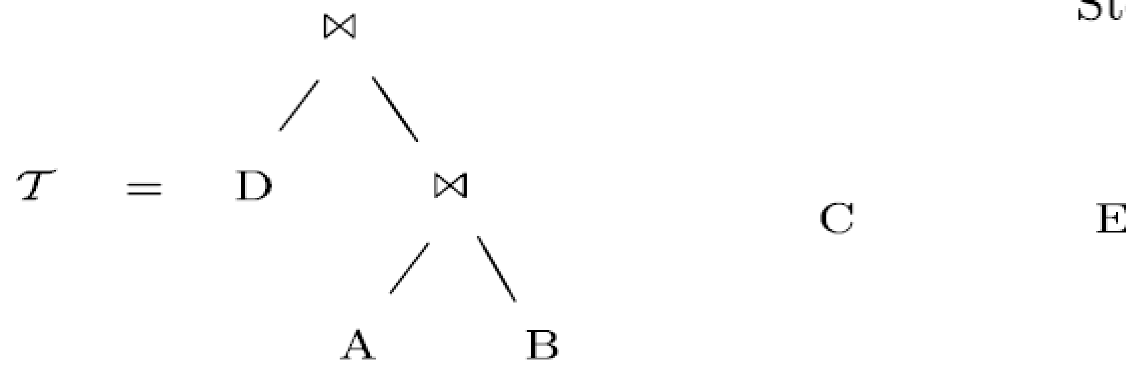


(eval = 300) (eval = 13) (eval = 2387)

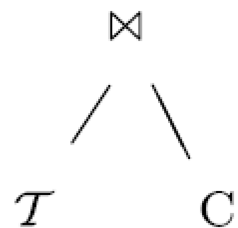
Step 3: 3-way joins

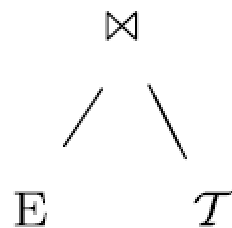
IDP: Optimizing an example query (cont'd)

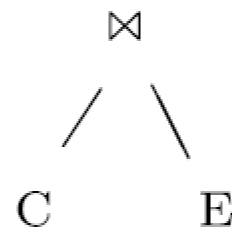
Step 4: select subplan;
start 2nd iteration



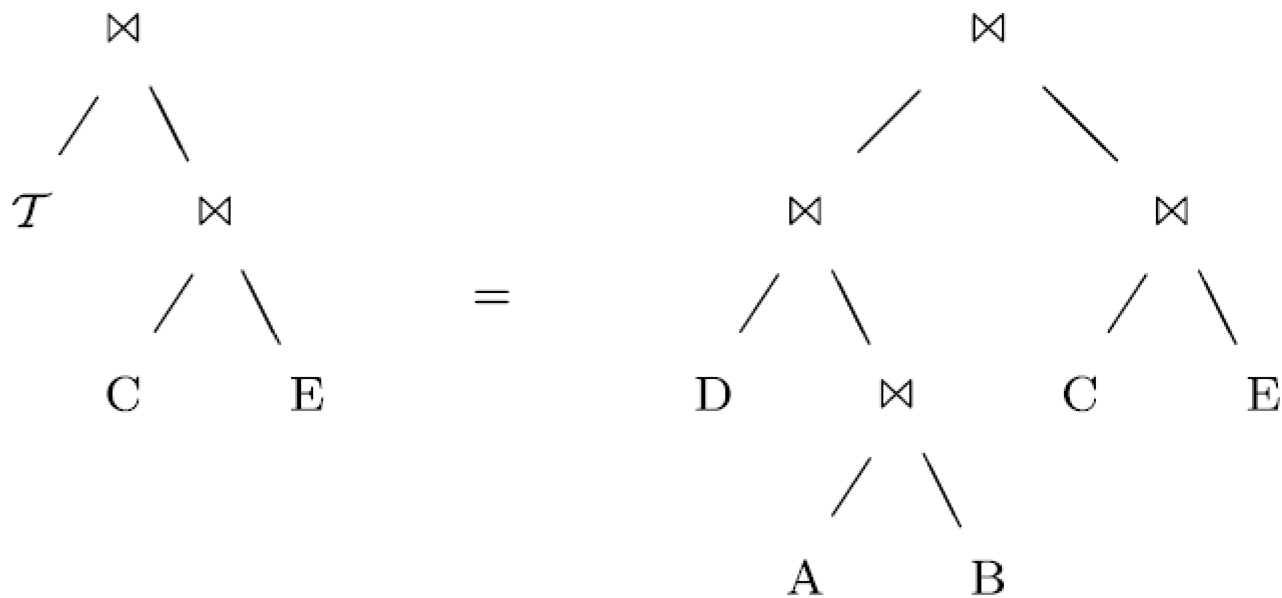
Step 5: 2-way joins;
2nd iteration







IDP: Optimizing an example query (cont'd)



Approaches to reduce DP search space

- Rank based Pruning ([3])
  - For handheld devices, highly memory constrained environments
  - Sort all the join candidates by plan cost and keep the cheapest candidate
  - Completely greedy, low quality plans are generated

- Avoiding cross products ([22])
  - Csg – connected subgraph sets of a query graph identified using BFS
  - Cmp –complementary subgraphs – Non overlapping subgraphs with a given subgraph
  - Ccp - Csg-cmp pairs / Connected complementary pairs are identified
  - Non overlapping relation pairs that contribute to subquery plan search space

- Left deep trees
  - Memory constrained environments like in [3] use them instead of bushy trees

- In [36], Vance and Maier proposed inclusion of Cartesian products and bushy trees. [24] encourage bushy trees in Star burst optimizer but keep a bound (Parameterized search space enumeration)

Approaches to reduce DP search space (cont'd)

- Multi-query Optimization (reuse during processing)
  - Identifying common sub-expressions both intra query and inter query
  - Exactly same nodes in the expressions
  - Result reuse during query processing as against plan reuse during optimization

- Top Down query optimization
  - Top down way of generating join candidates as expressions
  - Find an optimal sub expression of a given expression as you go down the lattice
  - Split a query graph into two connected components finding minimum cut sets at every level ([5])
  - Memo table to share plans among different queries for common (exactly same) sub expressions
  - Logically equivalent sub expressions (various join orders of a candidate) are denied plan generation ([31])

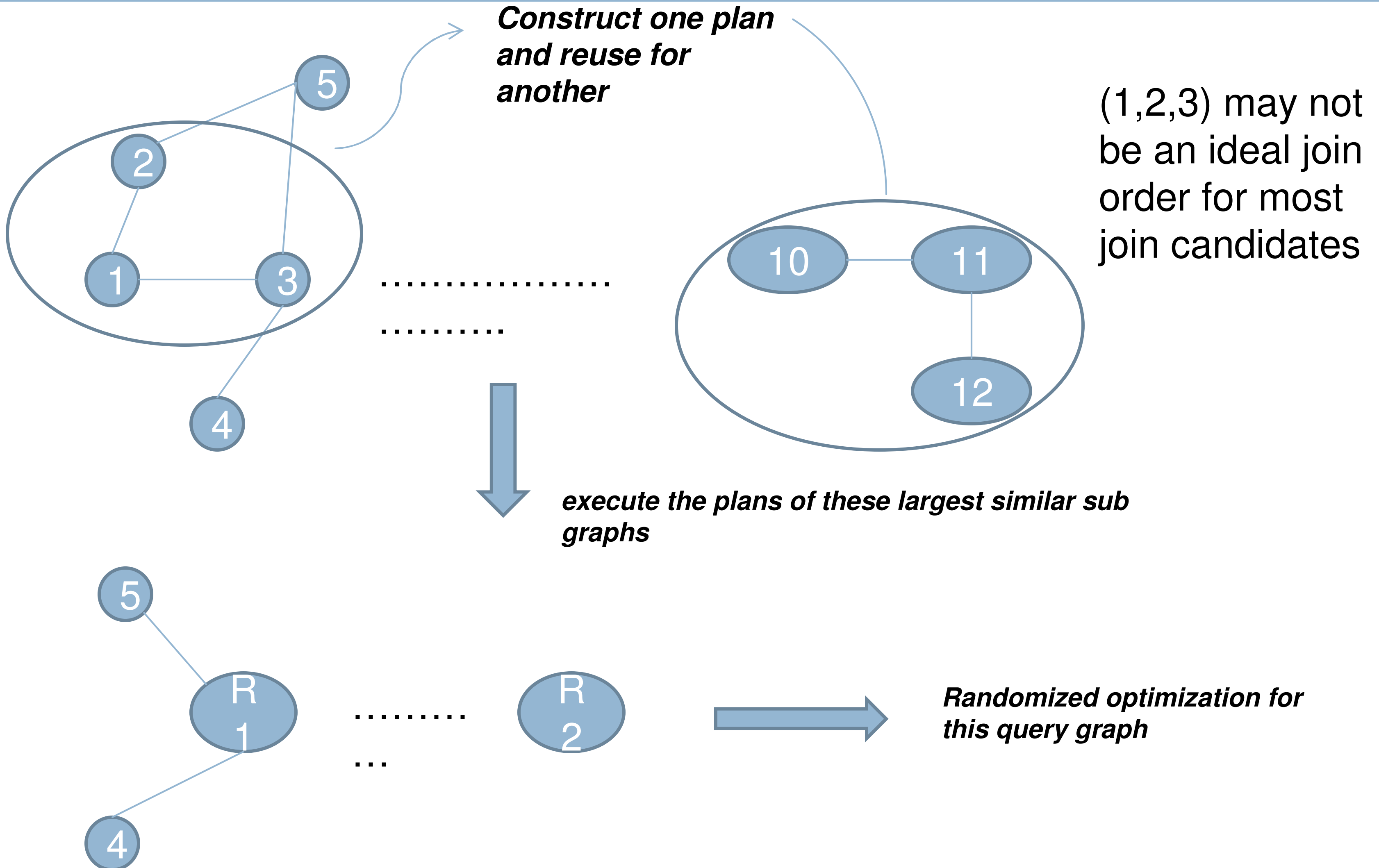
- Randomized query optimization
  - Moves in plan search space towards local minimum (Iterative Improvement)
  - Uphill moves allowed with reducing probability to avoid local minimum (Simulated Annealing)
  - 2 Phase Optimization – Runs Iterative improvement followed by Simulated Annealing
  - Tabu search keeps track of recent plans in a list to avoid getting trapped in cyclic moves
  - KBZ, AB algorithm- finds appropriate join sequence in linearized join trees from a spanning tree in join graph

- Genetic query optimization - Best plans are joined to generate join output
- Heuristic based optimization – Pushing selections down the join tree, early projections, late cross products, ordering relations in the order of increasing intermediate result sizes

Exploiting largest similar sub queries for query optimization

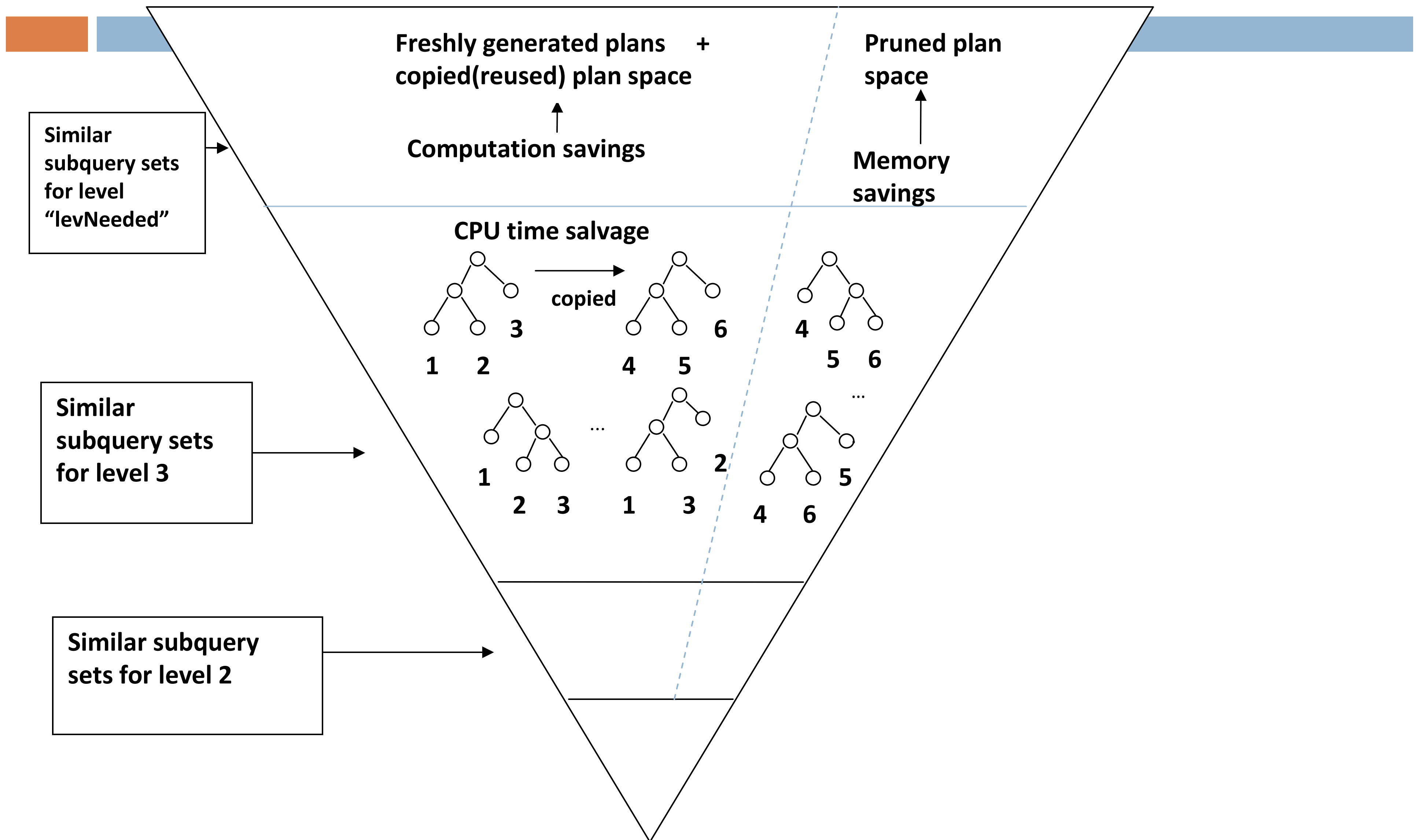
- [2] uses the notion of similar sub queries.
- Unlike common sub expressions, similarity allows for the expressions to contain different sets of tables.
- Based on relation size and selectivity (in essence similar result size) similarity is defined
- Largest similar subgraphs in a query graph are identified, plans are reused and executed
- Subgraphs replaced by result nodes
- Query optimization follows using a randomized algorithm like AB
- Basic drawback – Fixing join order prematurely (refer to the figure)

Exploiting largest similar sub queries for query optimization

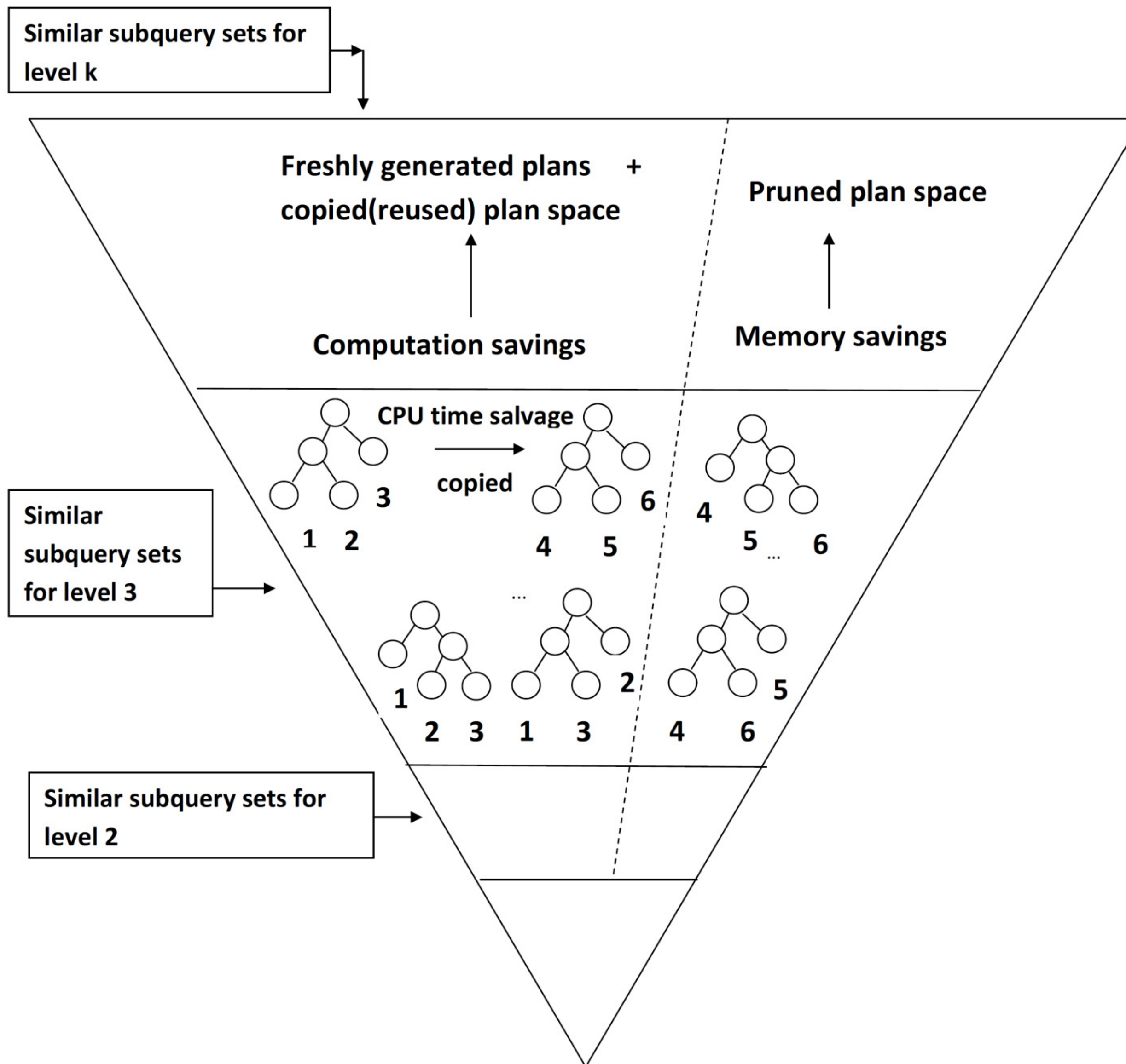


- Reuse of plans among similar sub queries in randomized scheme is not so beneficial (as we have seen, compromise in plan quality)
- What if re-use is applied to the join candidates in DP lattice?
- But only largest similar sub queries cannot yield any benefit
- So identify all sized similar sub queries in a given query
- Re-use plans among similar sub queries at all levels in the DP lattice
- Could be an alternative to Pruning

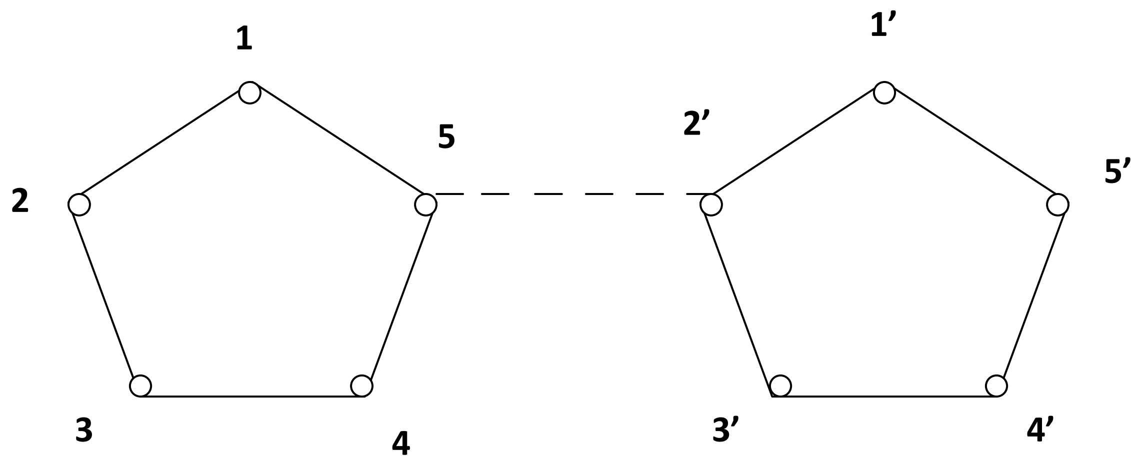
Basic working of our scheme



Basic working of our scheme



Basic working of our scheme (cont'd)



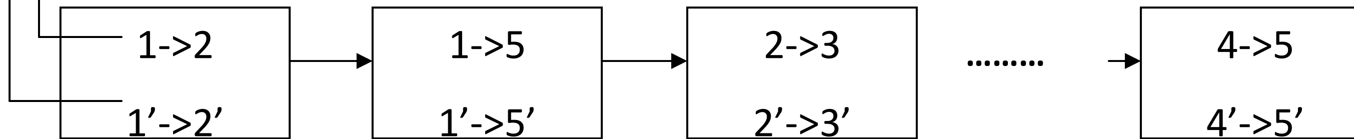
- [2]/ largest similar subgraph detection will reduce the two pentagons to two result nodes and continue AB/II on a query graph with two nodes connected by an edge.

- [3] / pruning based scheme will start hub detection from level 1, and continue.
  - 2' and 5 are the single rooted hubs. So at level 2, we have only one join candidate containing 2'
  - One more 2-level join candidate with 5 in it

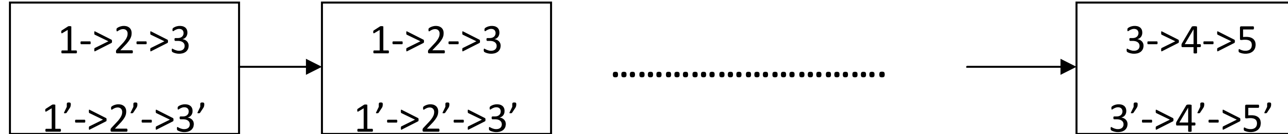
Basic working of our scheme (cont'd)

(1,2) similar to (1',2'), Reuse plans

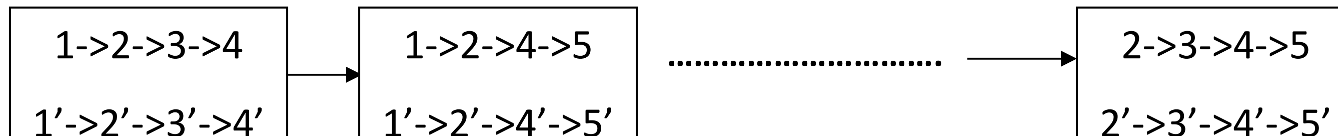
SETS for LEV-2



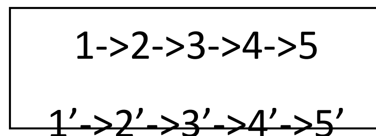
SETS for LEV-3



SETS for LEV-4



SETS for LEV-5



.....

- *Cover set constructed as per our scheme*

Our scheme involves two steps:

- Generation of the cover set of similar subgraphs from the query graph.
- Re-use of query plans for similar subqueries represented by the similar subgraphs.
- *Subgraph Reuse based DP (SRDP) and Subgraph Reuse based IDP (SRIDP)* arrive out of embedding our scheme in DP and IDP algorithms respectively.

Cover set of similar subgraphs

- Here "total" indicates the total number of similar subgraph sets at level "lev" in the DP lattice.
- Subgraphset*i* indicates the i th similar subgraph set.
- The summation or total collection of all such subgraph sets at level "lev" is represented by $Sets_{lev}$.
- The total collection of all such subgraph sets over all levels gives the cover set of subgraphs.

$$\sum_{lev=2}^{n} Sets_{lev} \text{ where } Sets_{lev} = \sum_{i=1}^{total} Subgraphset_i$$

Generating Cover set of similar subgraphs

- Common subgraphs: Isomorphic subgraphs having identical structure and features

- Similar subgraphs: Isomorphic structure and "*similar*" features

- Similarity in this context means the value of features should lie within the specified threshold.
  - Feature 1: Relation size , Feature 2: Selectivity

- A pair of similar subgraphs *{S, S'} is defined as a pair of subgraphs having the* same graph structure and *similar features, i.e, each vertex, v, in S should have* a corresponding vertex, v', in S' such that differences between table sizes and selectivities of the containing edges lie within the corresponding error bounds.

- If similar subgraphs of all sizes are found in a given query graph it is termed "Cover set"

Generating Cover set of similar subgraphs

Generation of cover set involves two stages:

1 a. Formation of seed list

- grouping base relations (vertices) in the query graph based on the difference in their relation size and presence of indexes

1 b. Growth of seed list

-base relations are paired to form 2-sized subgraph sets.

2. Growth of “lev” sized similar subgraph sets to obtain “lev+1” sized sets.

Stage 2 is run iteratively till we can no longer find similar subgraph sets of an extended size.

Construction of Seed List

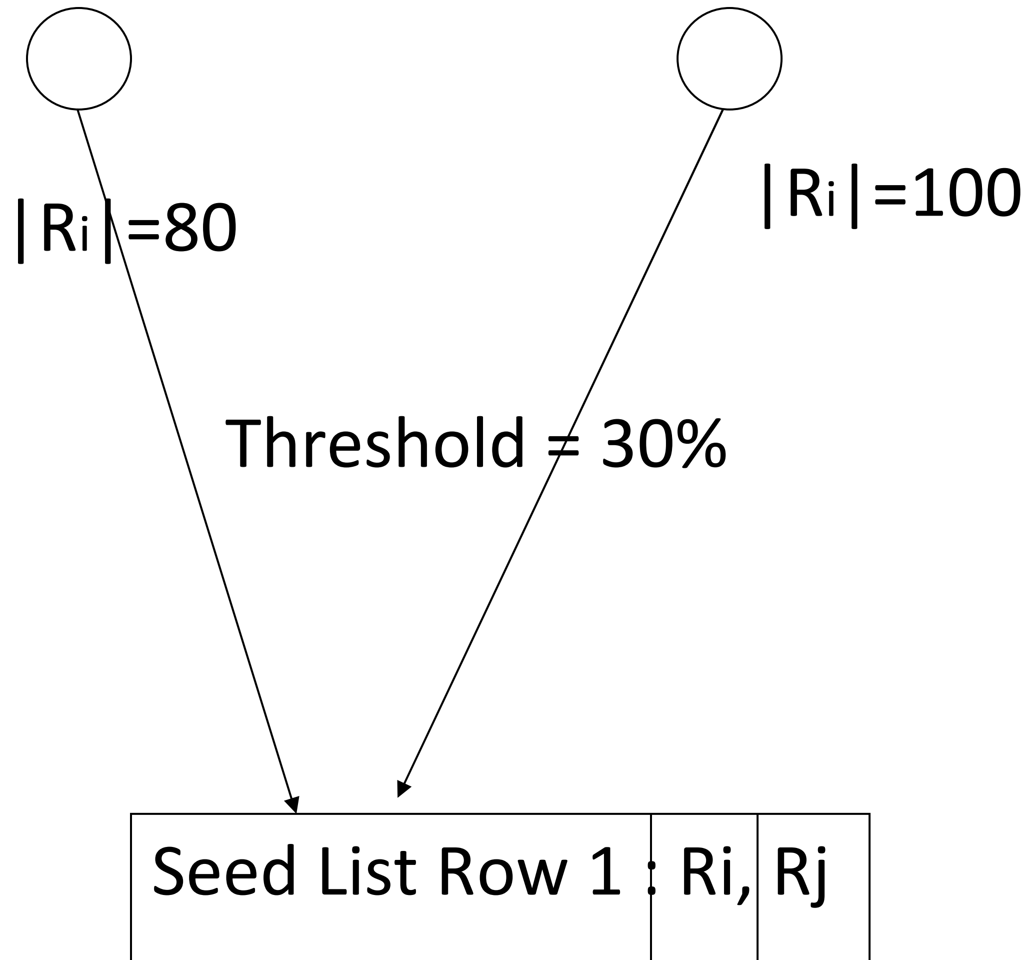
- Two relations fall into the same seed group if their relation sizes are within the relation size relaxation (threshold)

$$\frac{|relSize(R_i) - relSize(R_j)|}{max(relSize(R_i), relSize(R_j))} < relErrorBound$$

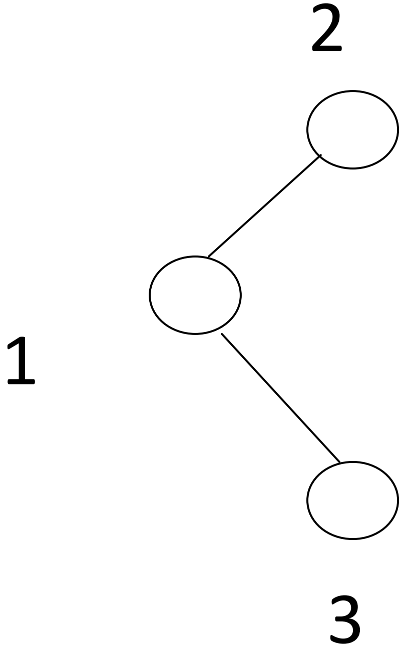
and if Ri and Rj are similar with respect to indexes. Either both of them should be indexed or both should have any indices built upon them.

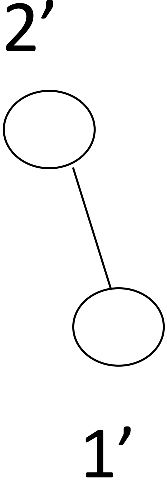
GroupId	Seeds
0	1,1'
1	2, 2'
2	3, 3'
3	4, 4'
4	5, 5'

Construction of Seed List (cont'd)



Growth of seed list





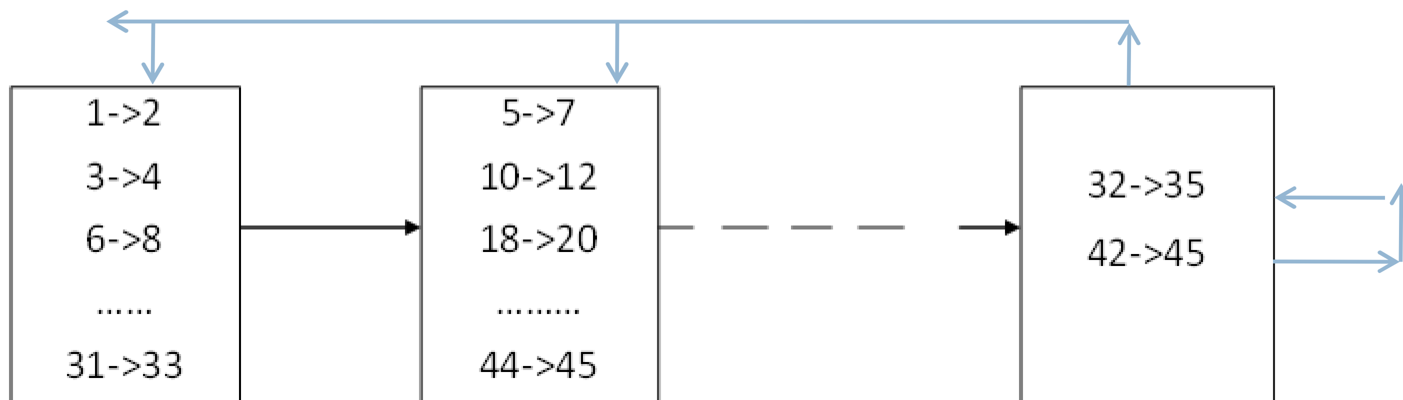
1->2

1->3

1’->2’

Growth of seed list (cont'd)

- Limited Scanning
- First member alone of a subset box to scan the current level's set



Growth of seed list (cont'd)

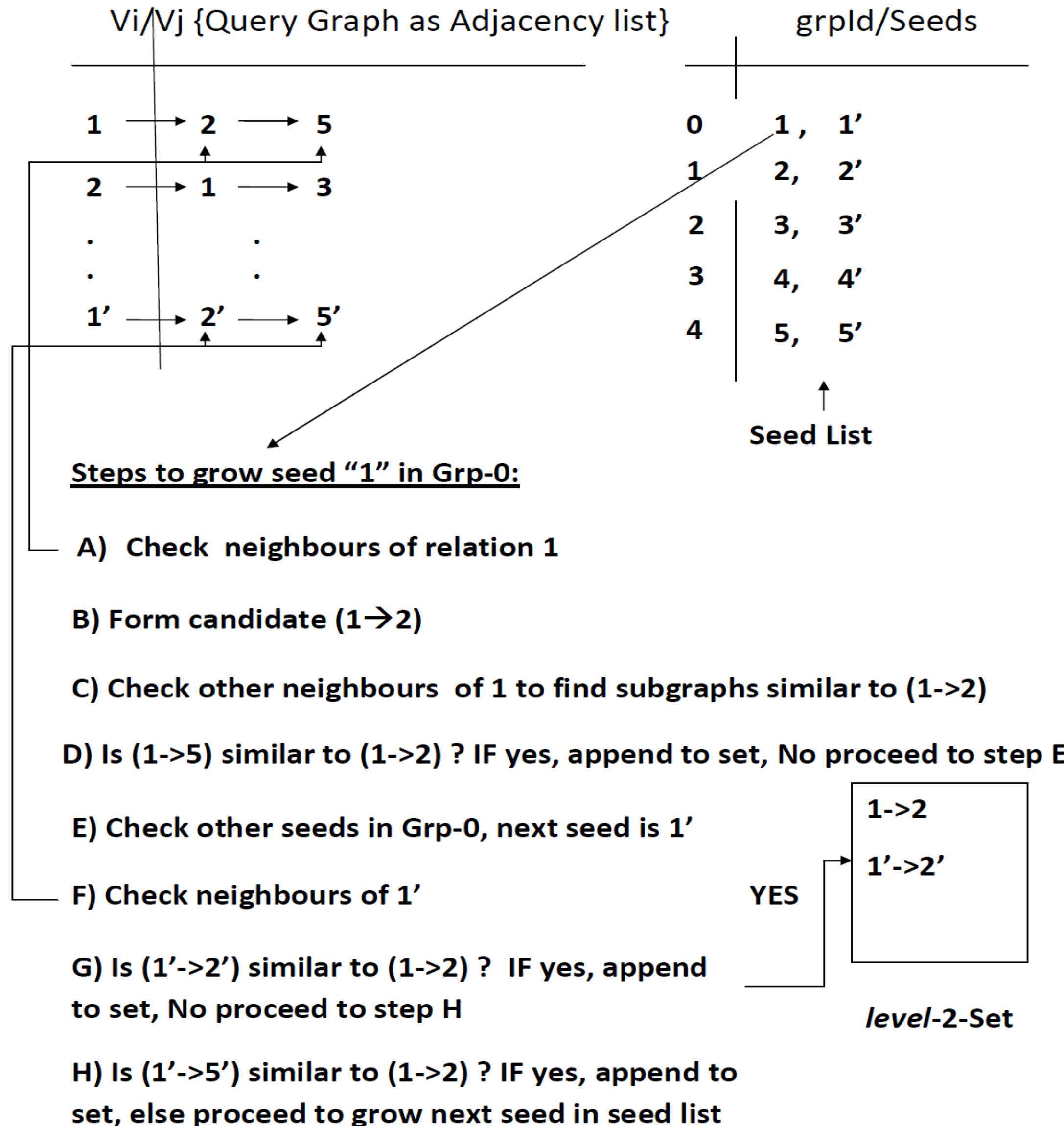
- Growing a seed means growing a base relation into a 2-sized subgraph.
- By default, we have nC_2 pairs of size 2.
- But they may contain some cross products. We generate only connected subgraphs.

- For this, we refer to the adjacency list and pair the given seed "S" with each of its adjacent vertices in the query graph.

- Few similar subgraphs are obtained by pairing "S" with different vertices directly connected to it. For example, (1,2) may be similar to (1,4) if 2 and 4 are within size threshold and selectivities of these edges are within similarity threshold.

- To get similar subgraphs, we scan the seed list and grow other members from the group of "S" to 2-sized subgraphs.
  - Because a group mate of "S" is a similar node. Growth of a similar node can fetch similar subgraphs

Growth of seed list (cont'd)



Growth of sub graphs

Two edges are similar if the participating nodes are similar with respect to index presence and have their table sizes differing within *relErrorBound and selectivity* difference between the predicates is within *selErrorBound i.e,*

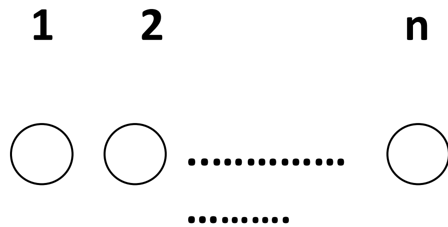
$$\frac{|sel(R_i)-sel(R_j)|}{max(sel(R_i),sel(R_j))} < selErrorBound$$

Growth of a sub graph is same as growth of a seed except that here, we have a collection of relations to grow.

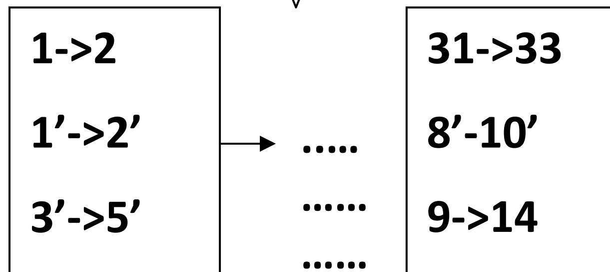
So grow every vertex in the sub graph of size “lev” to obtain candidate subgraphs of “lev+1”

Growth of seeds versus growth of subgraphs

Growing a seed List



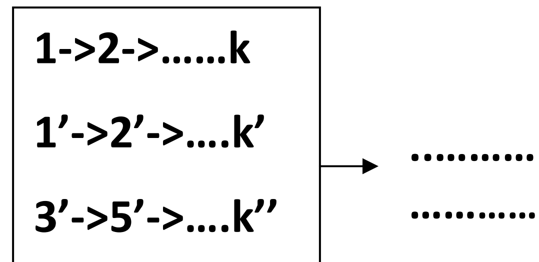
Seeds



2-sized graph sets

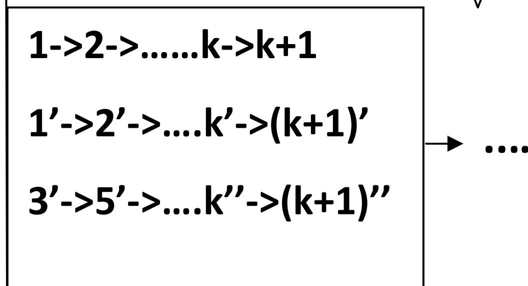
Sets*2*

Growing sets of similar subgraphs



k-sized graph sets

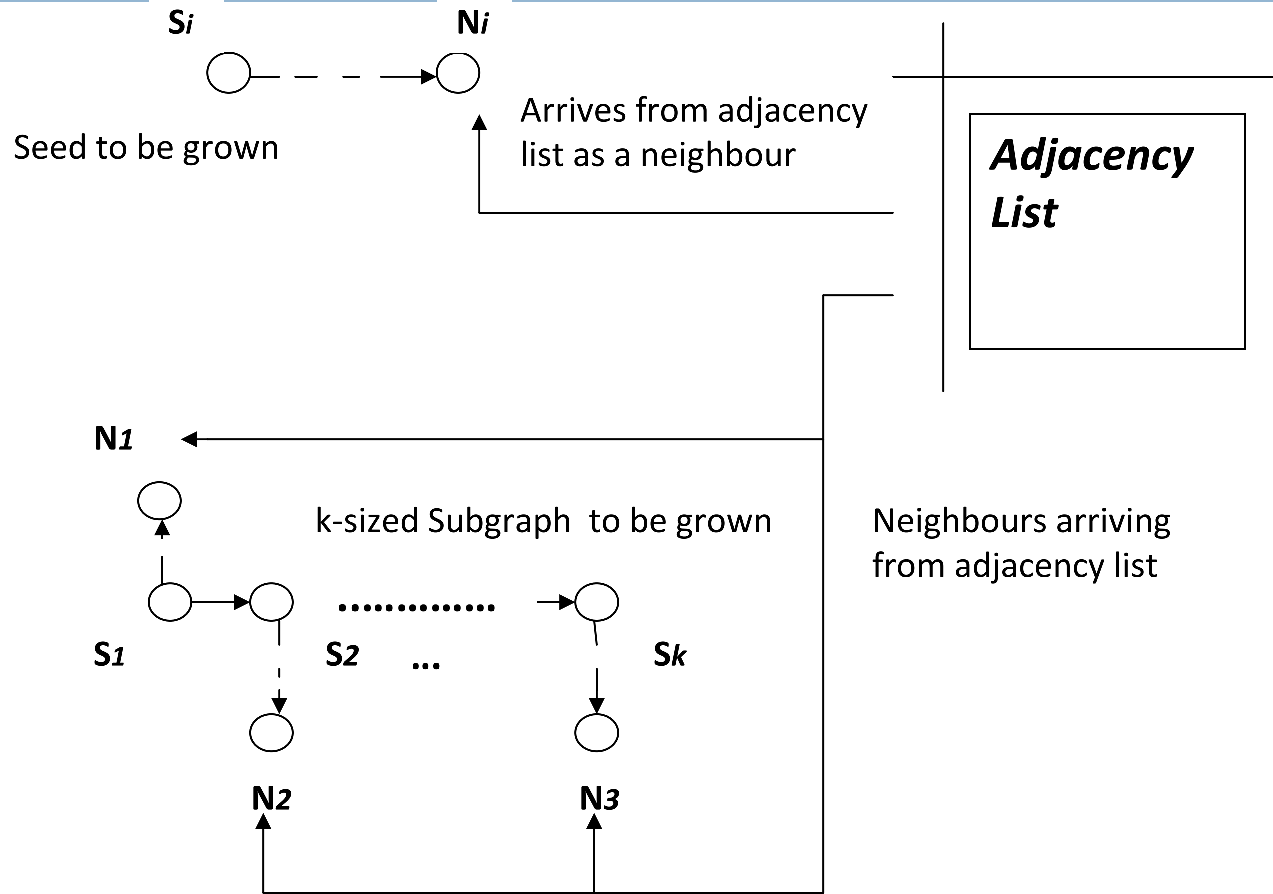
Sets*k*



K+1-sized graph sets

Sets*k+1*

Growth of seeds versus growth of subgraphs (cont'd)



Plan generation using similar sub queries

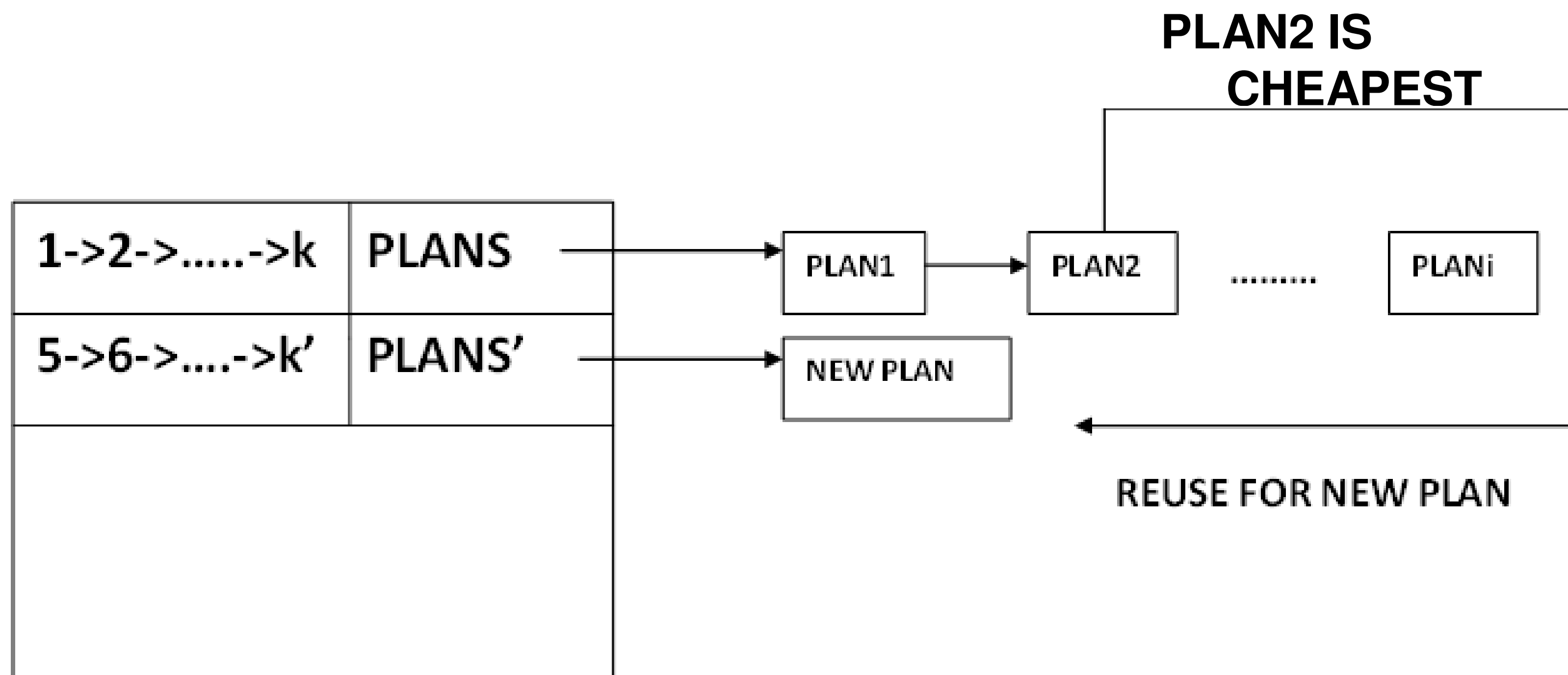
- The similar subgraph sets also hold bitmaps for each set for faster comparison and a pointer to the set of plans for all the join candidates corresponding to the constituent subgraphs.

- To generate a plan for a join candidate “j”,
  - Check if it has a corresponding sub graph entry in the cover set.
  - If yes, check if any of the subgraph’s companions in the set already have their plans built.

    -If yes, construct a plan for “j” by reuse.

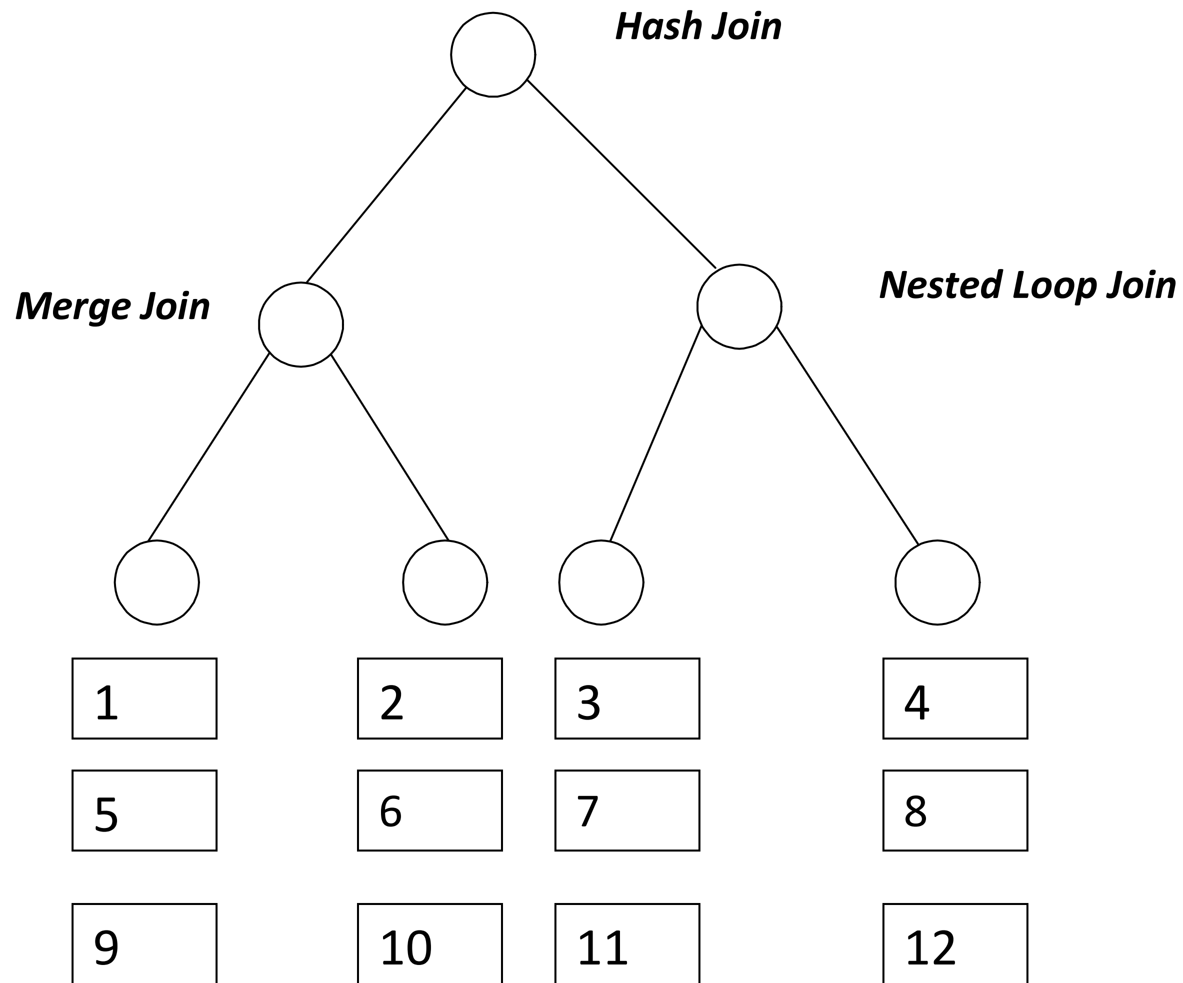
    -If no, build a fresh plan and make an entry of the pointer in the subgraph set
  - If no, build a fresh plan for “j”

Plan generation using similar sub queries (cont'd)

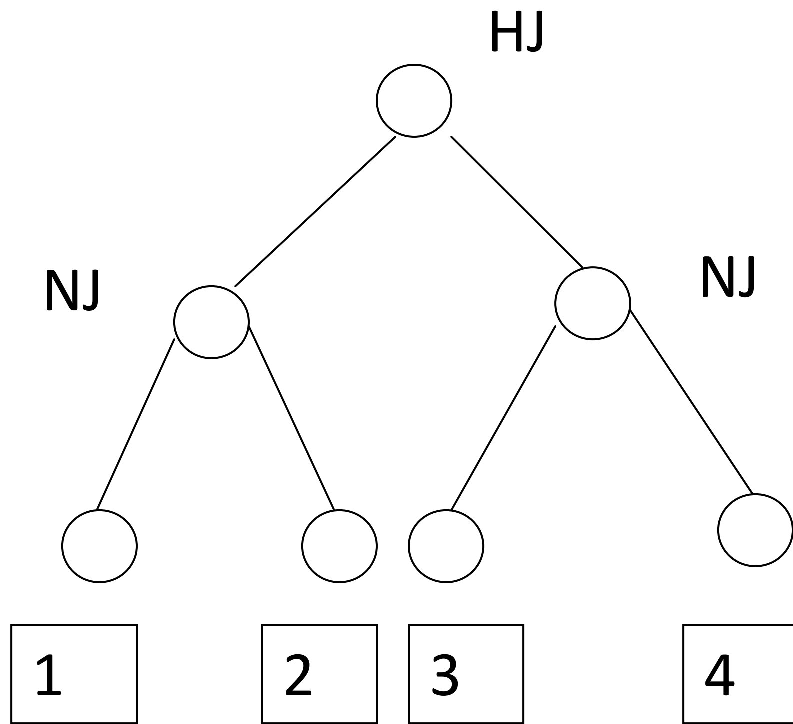


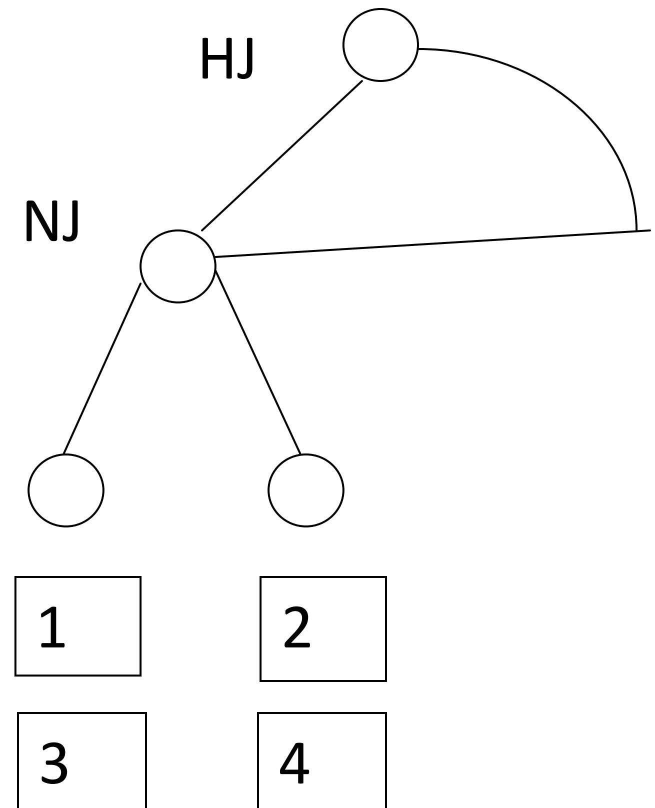
Memory savings arrive from not building plans for various join orders of (5,6,..,k') because PostgreSQL holds all of them in main memory

Intended Plan Reuse



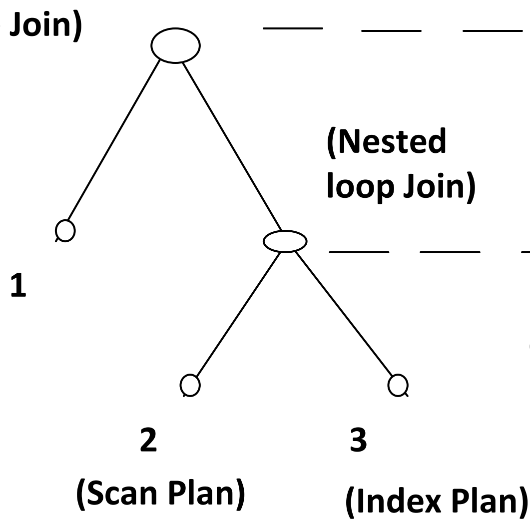
Intended Plan Reuse (cont'd)

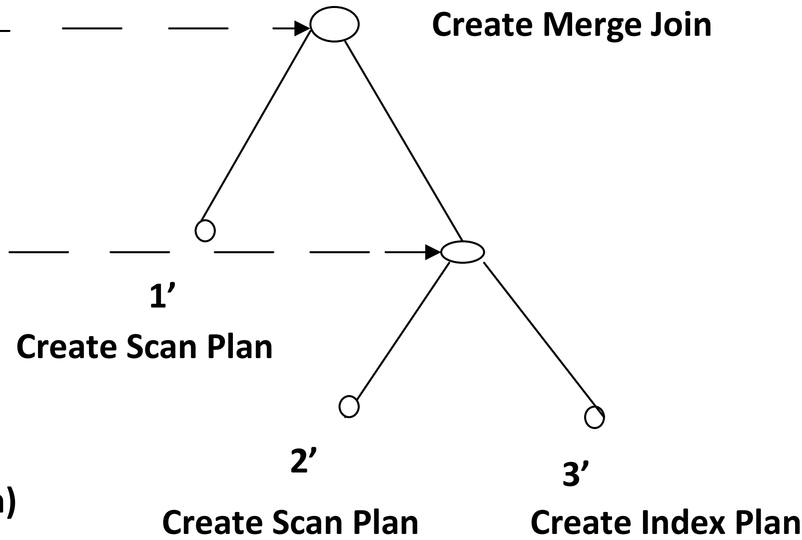


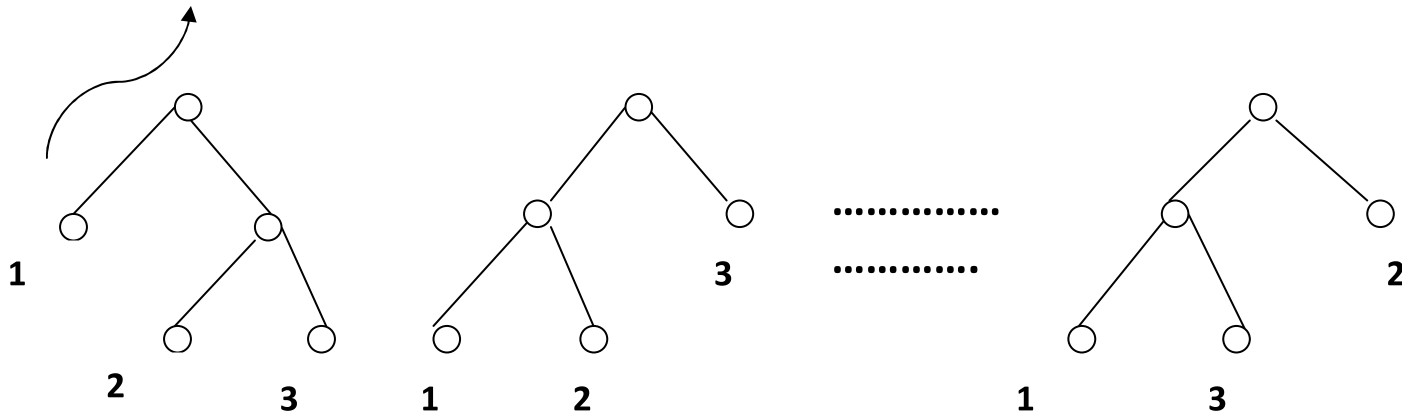


Plan Reuse done in PostgreSQL

Cheapest Plan selected





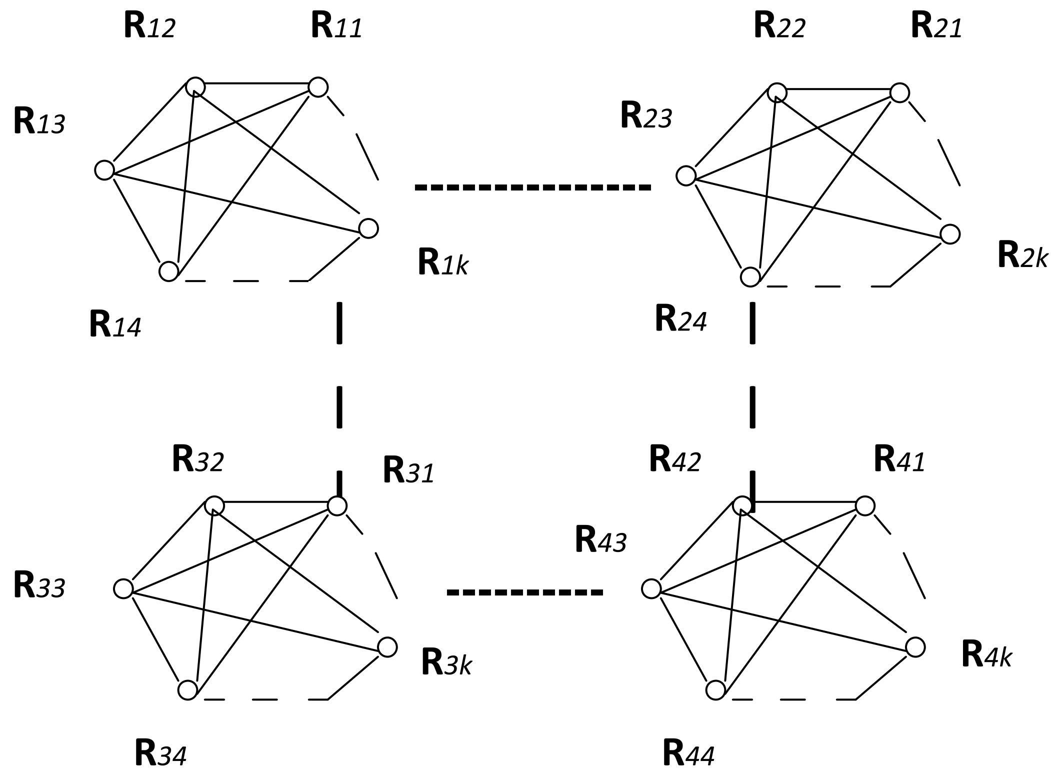


- *Plan reuse done as per our scheme*

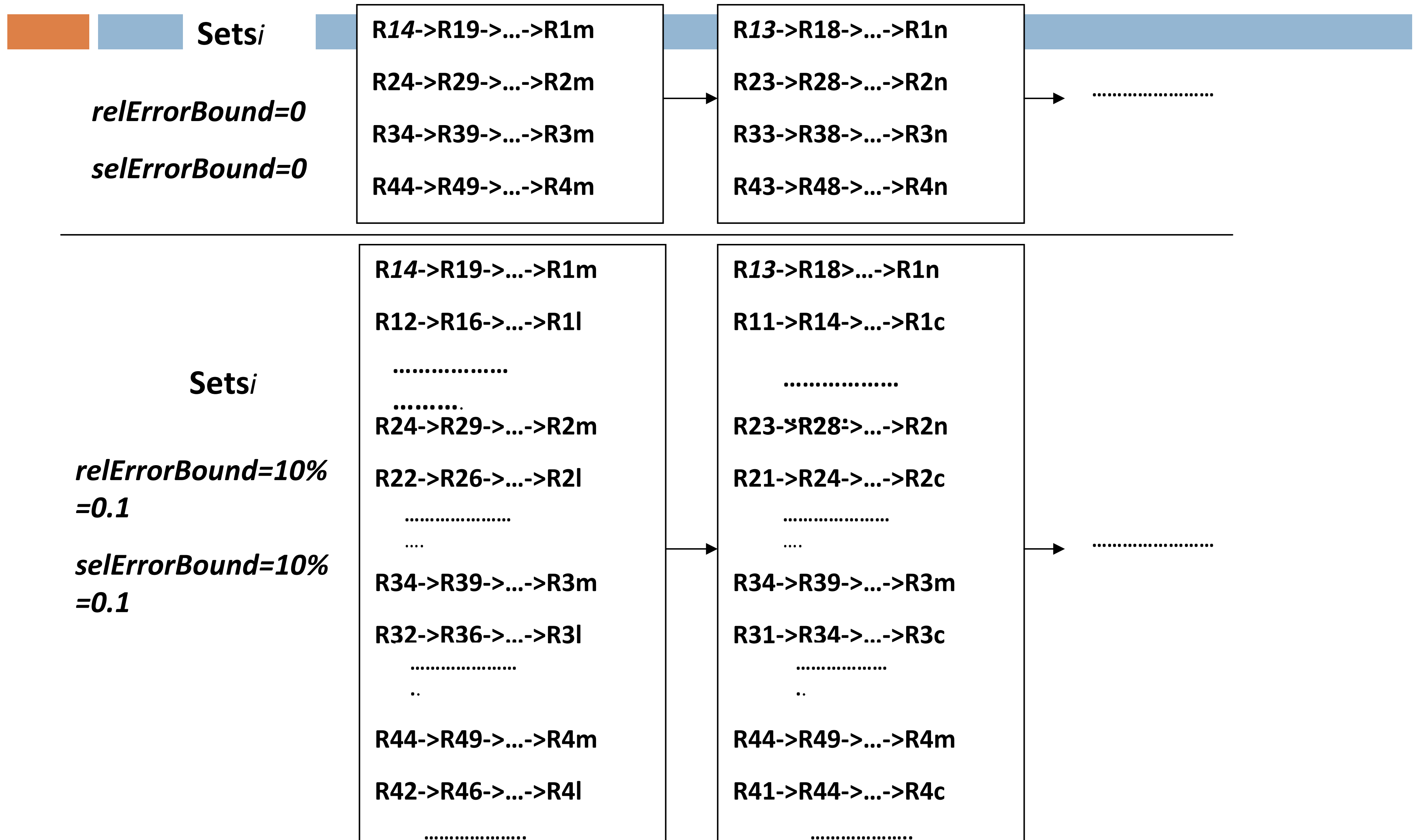
Making Cover set generation memory efficient

- For dense graphs, with increased relaxation of selectivity and table size difference, the number of similar subgraph entries in each set increase heavily
- If we reduce the fraction of generated subgraph sets, we may lose the opportunity of plan reuse for those sub queries but we do not compromise plan quality.
- The price we pay is extra plan generation but it is worthwhile given the savings in time and memory.
- We introduce a size based pruning of similar subgraph sets. All subgraph sets with a population less than a threshold fraction of the highest populated set will be pruned.
- If Prune factor = 1/j , allowed strength = largest strength / j

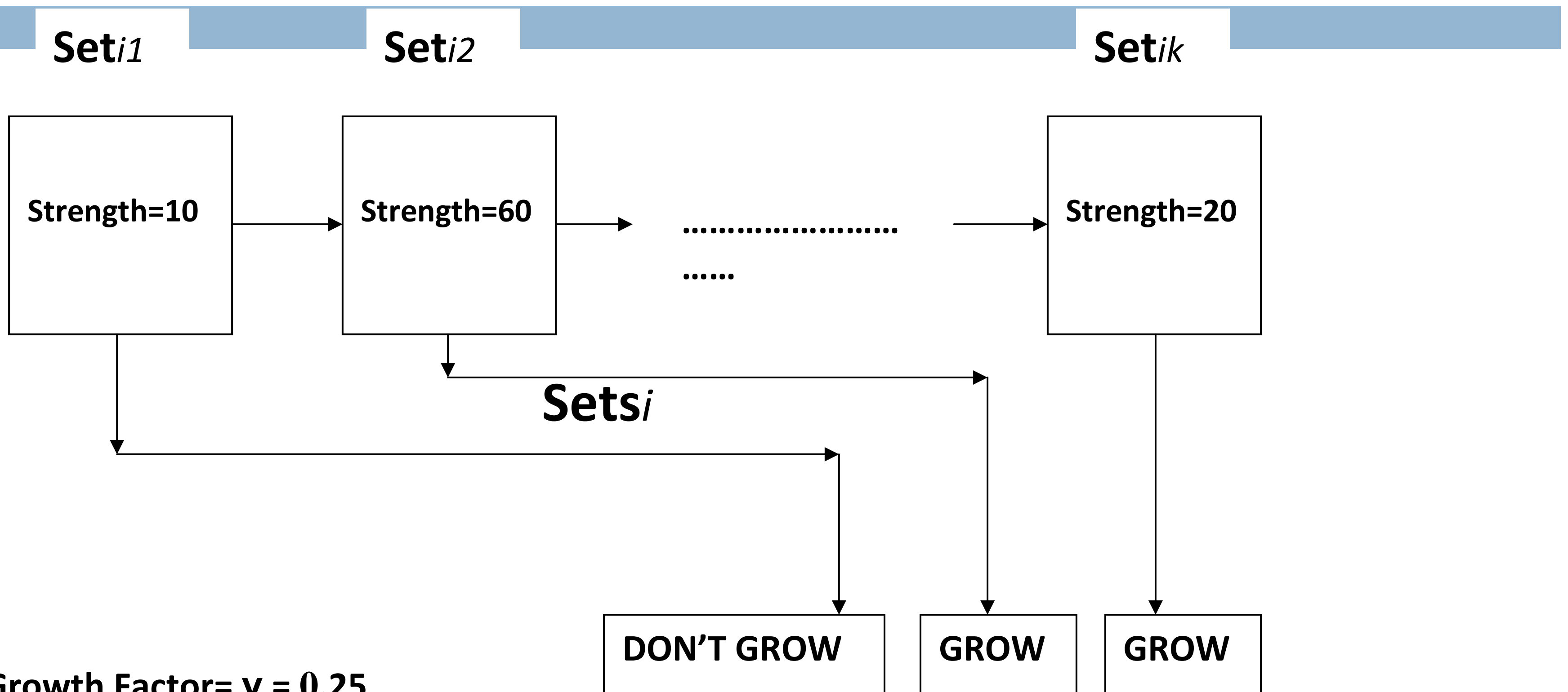
Increase in subgraph set population with error bound relaxation



Increase in subgraph set population with error bound relaxation



Making Cover set generation memory efficient (cont'd)



Growth Factor= γ = 0.25

Maximum strength= 60

Grow only if strength > = (γ*Maximum strength)

Improving memory efficiency of plan generation

- Cover set competes with sub query plans for main memory
- Avoid constructing the entire cover set at one go.
- Growth of sub graphs happens only on need, and the sub graph sets are deleted when
  they are no longer required
- In the Dynamic programming lattice, (SRDP / SRIDP)
  - Construction of plans for lattice level “k”
    - Grow level “k-1” sub graph sets to build level “k” sub graph sets
    - Delete “k-1” sub graph sets (seed list if k=2)
    - Build query plans referring to $Sets_{k+1}$

Performance Study

- The experiments were run on a PC with Intel(R) Xeon(R) 2.33GHz CPU and 3GB RAM. All the algorithms were implemented in PostgreSQL 8.3.7

- 80 tables database
- 30 columns
- 1000 to 8,000,000 tuples

- Default Parameters

Density Level	Similarity relaxation	prune factor
2	30,30	30

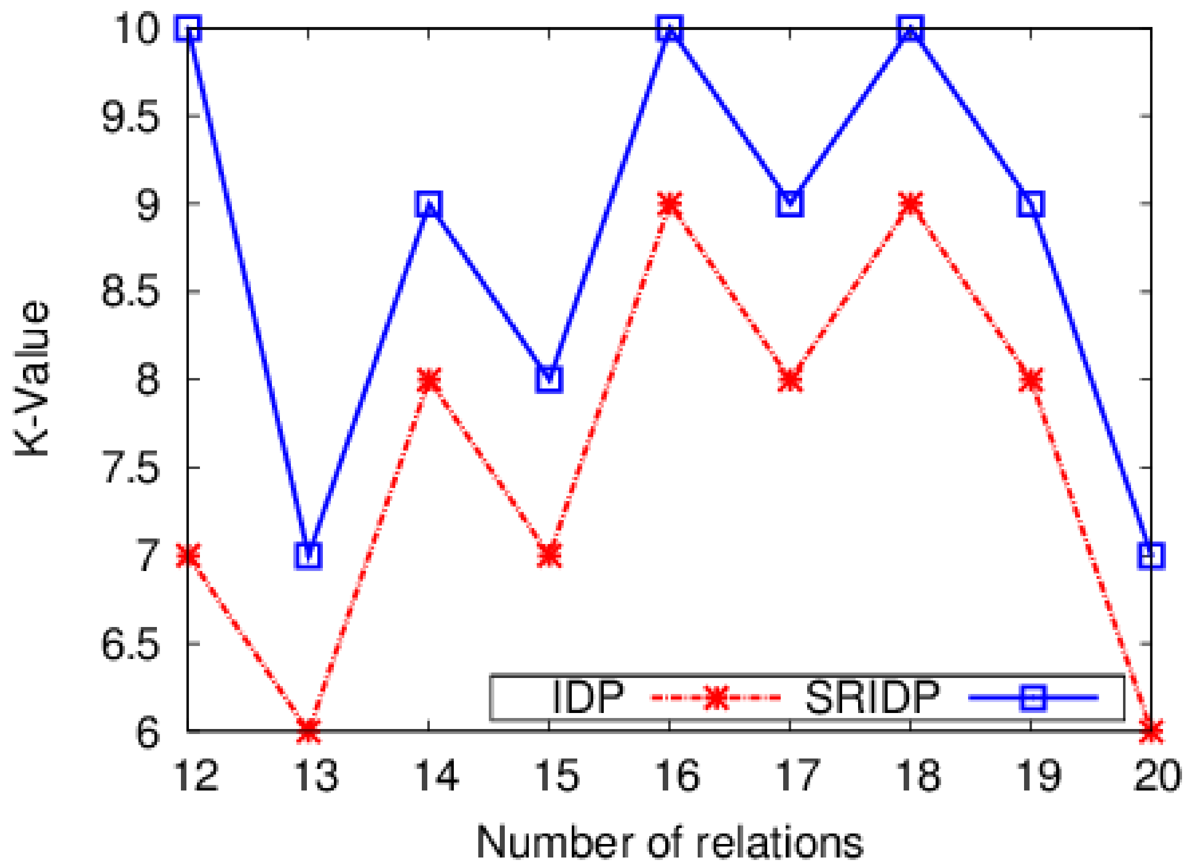
Density level k implies during query construction,
maximum allowed degree of a vertex = total number of vertices in the graph / k

Dense queries: Inferred Predicates (cont'd)

- Query Q5 in TPC-H benchmark
- "s_nationkey" is multi-referenced

```
select n_name, sum(l_extendedprice * (1 - l_discount)) as
revenue from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey and l_orderkey = o_orderkey and
l_suppkey = s_suppkey and
c_nationkey = s_nationkey and s_nationkey = n_nationkey
and n_regionkey = r_regionkey and
r_name = '[REGION]' and o_orderdate >= date '[DATE]' and
o_orderdate < date '[DATE]' + interval '1' year group by
n_name order by revenue desc;
```

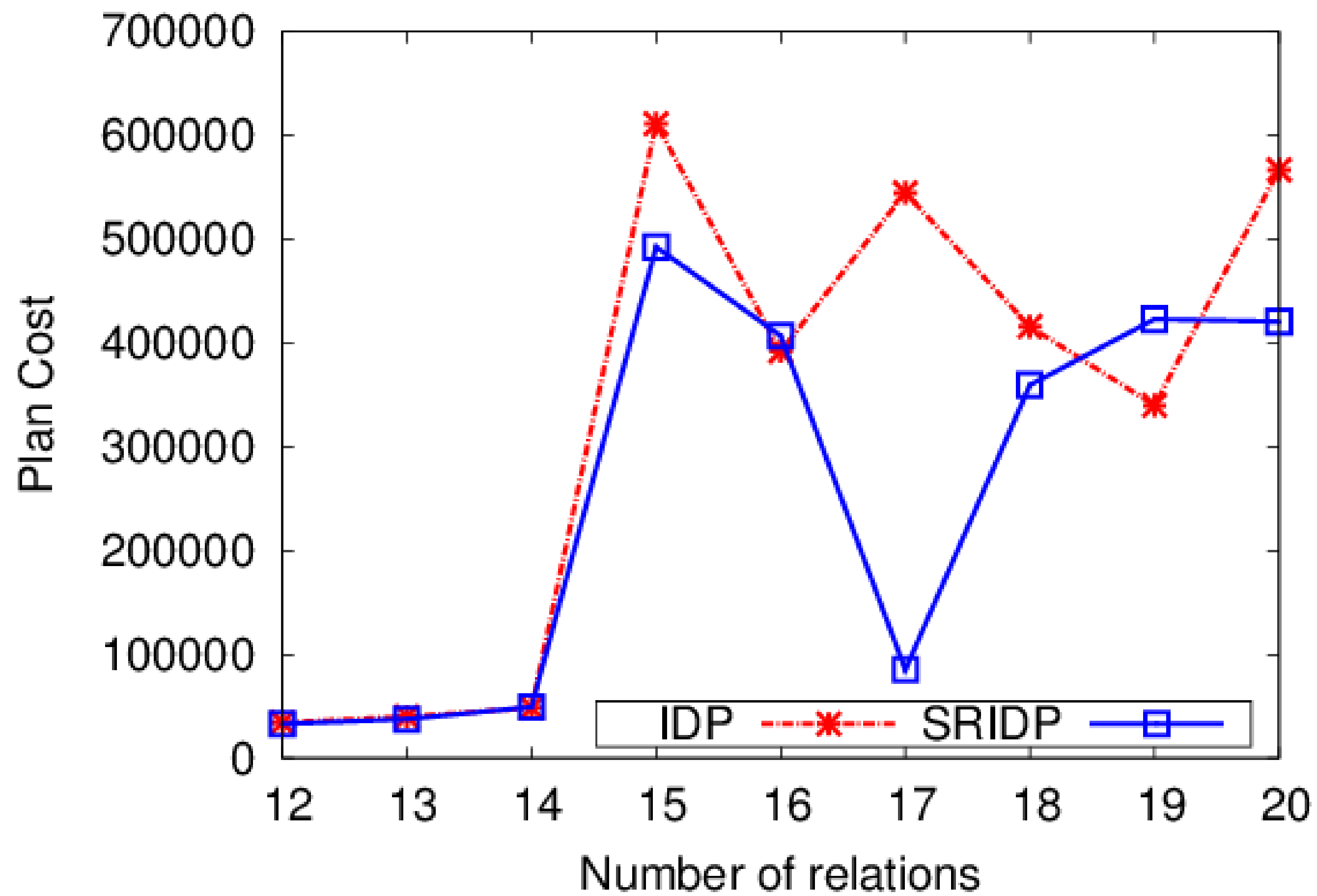
Varying number of relations in the query



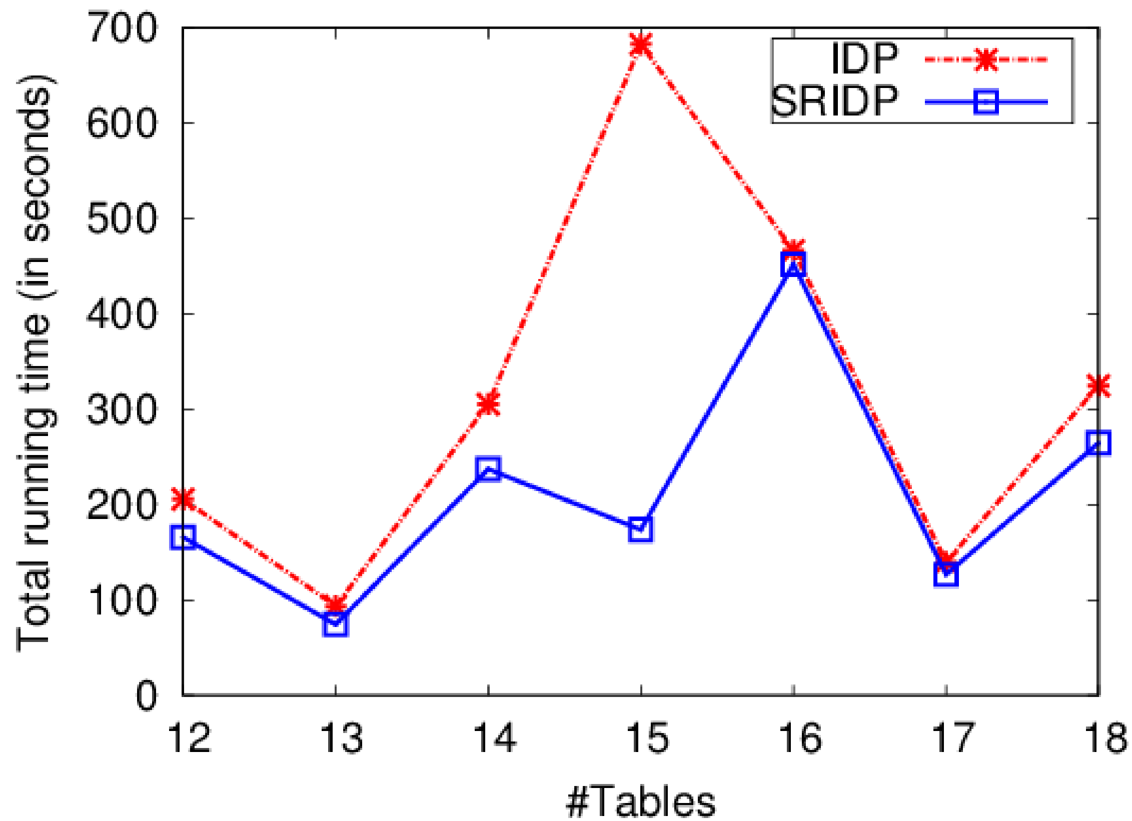
Actual cost measurements for density level 2

Number of relations	Plan cost IDP	Plan Cost SRIDP	Skyline DP(pruning)
12	34571.29	33174.26	out of memory
13	40316.21	37849.12	out of memory
14	48755.22	49652.81	out of memory
16	392045.96	407081.06	out of memory
17	544761.71	85452.12	out of memory
18	415910.6	359533.91	out of memory
19	340097.2	422955.86	out of memory
20	566904.82	420731.09	out of memory

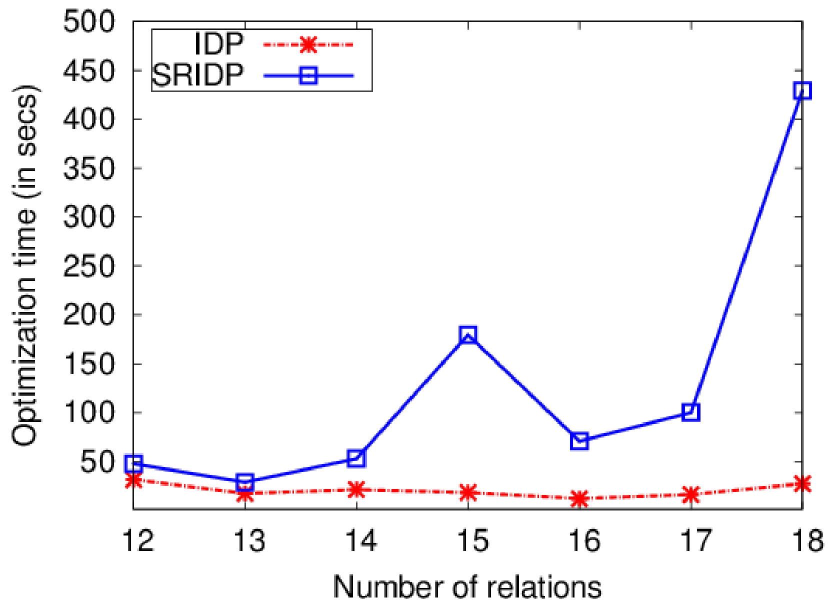
Varying number of relations for density level 2

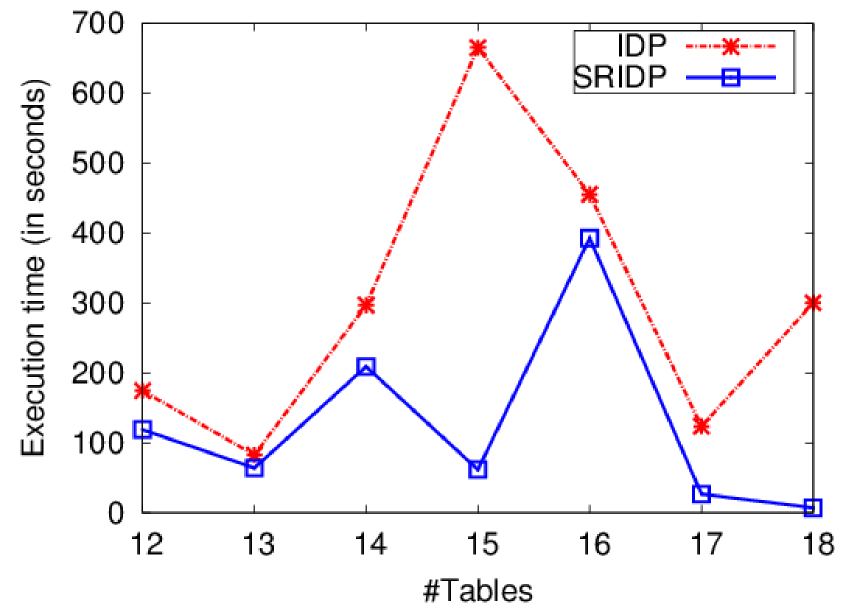


Measuring Total Running Time (opt+exec) for density level 2



Optimization time and Plan Execution time vs # relations for density level 2

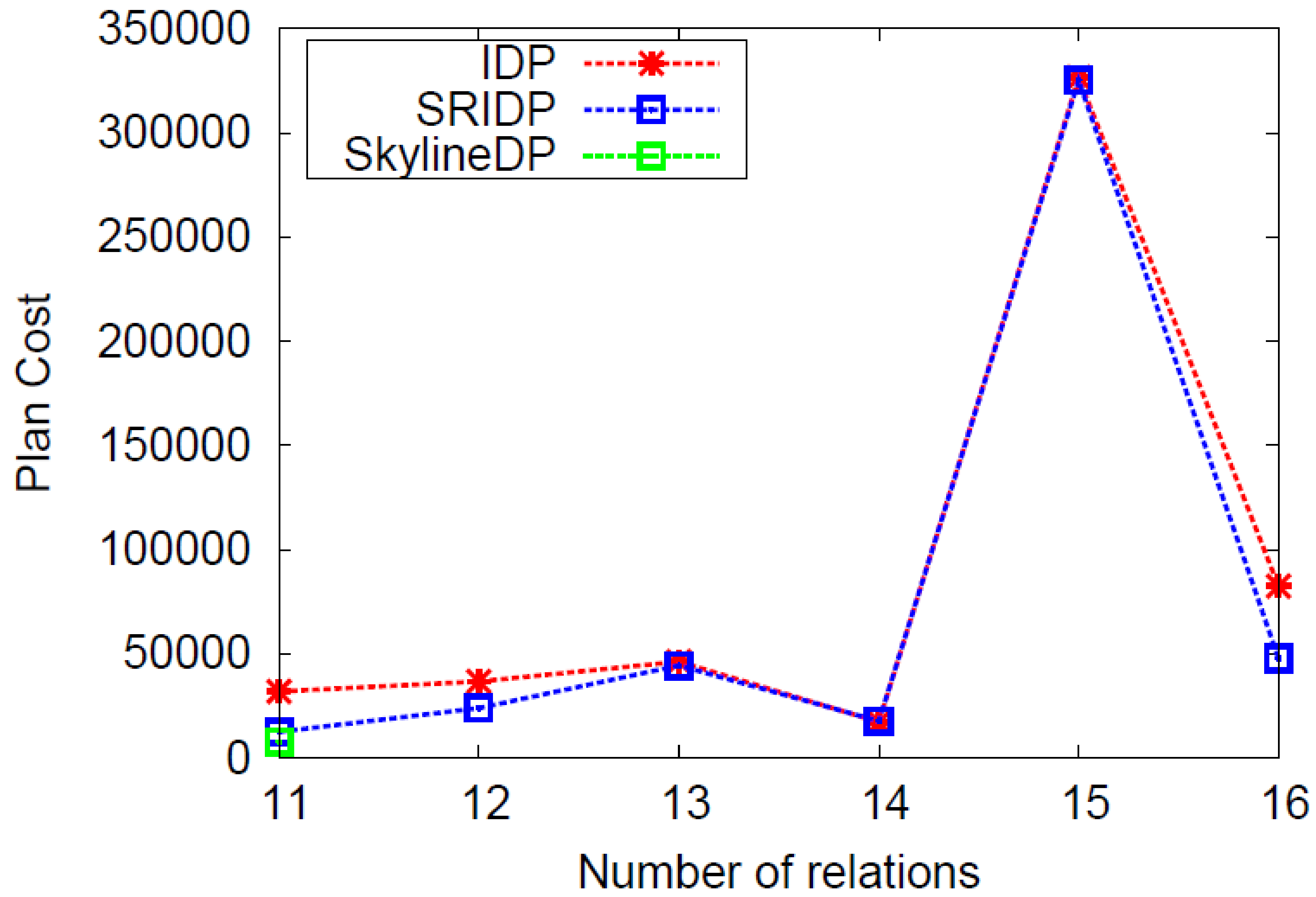




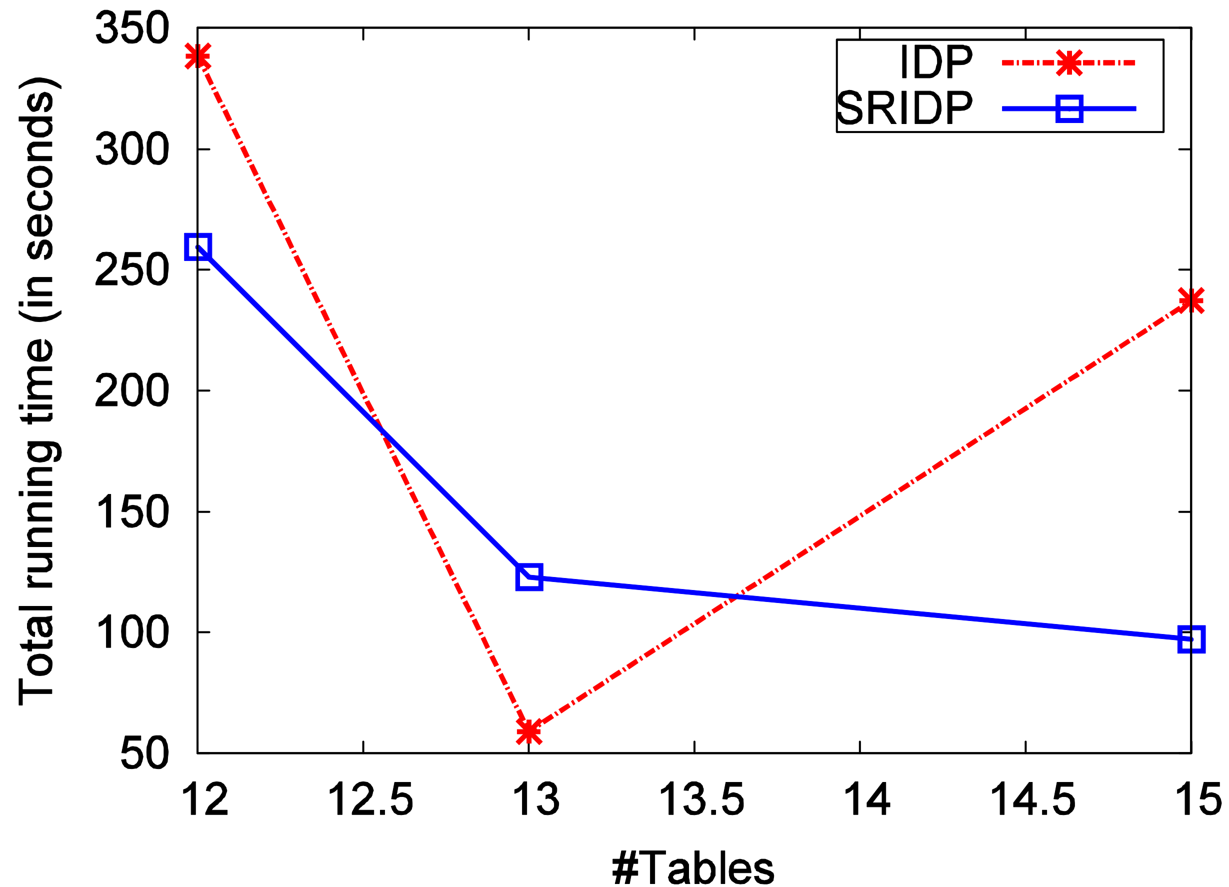
Actual cost measurements for density level 1

Number of relations	Plan cost IDP	Plan Cost SRIDP	Skyline DP(pruning)
11	31903.28	12486.45	7729.86
12	36729.46	23866.3	out of memory
13	46183.5	44360.81	out of memory
14	17192.68	17714.26	out of memory
15	326531.96	325158.31	out of memory
16	82425.42	47818.18	out of memory

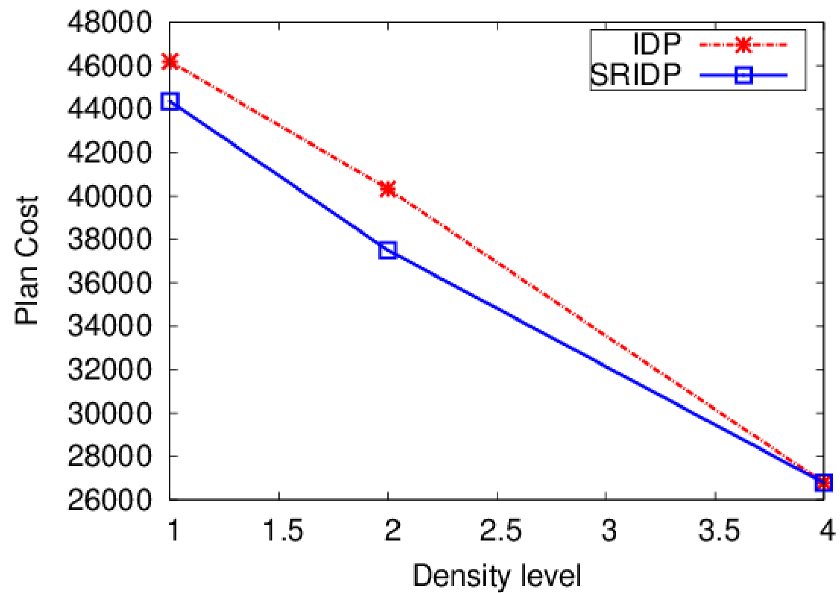
Varying number of relations for density level 1

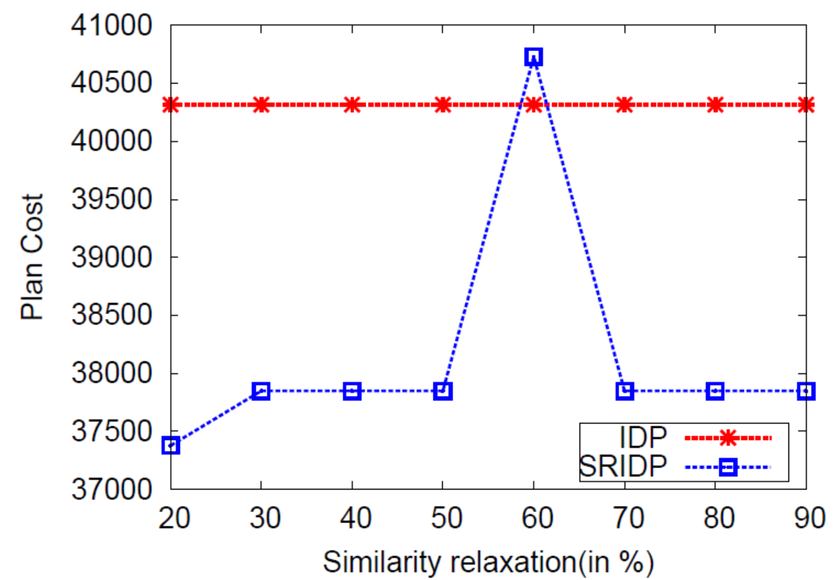


Measuring Total Running Time for density level 1

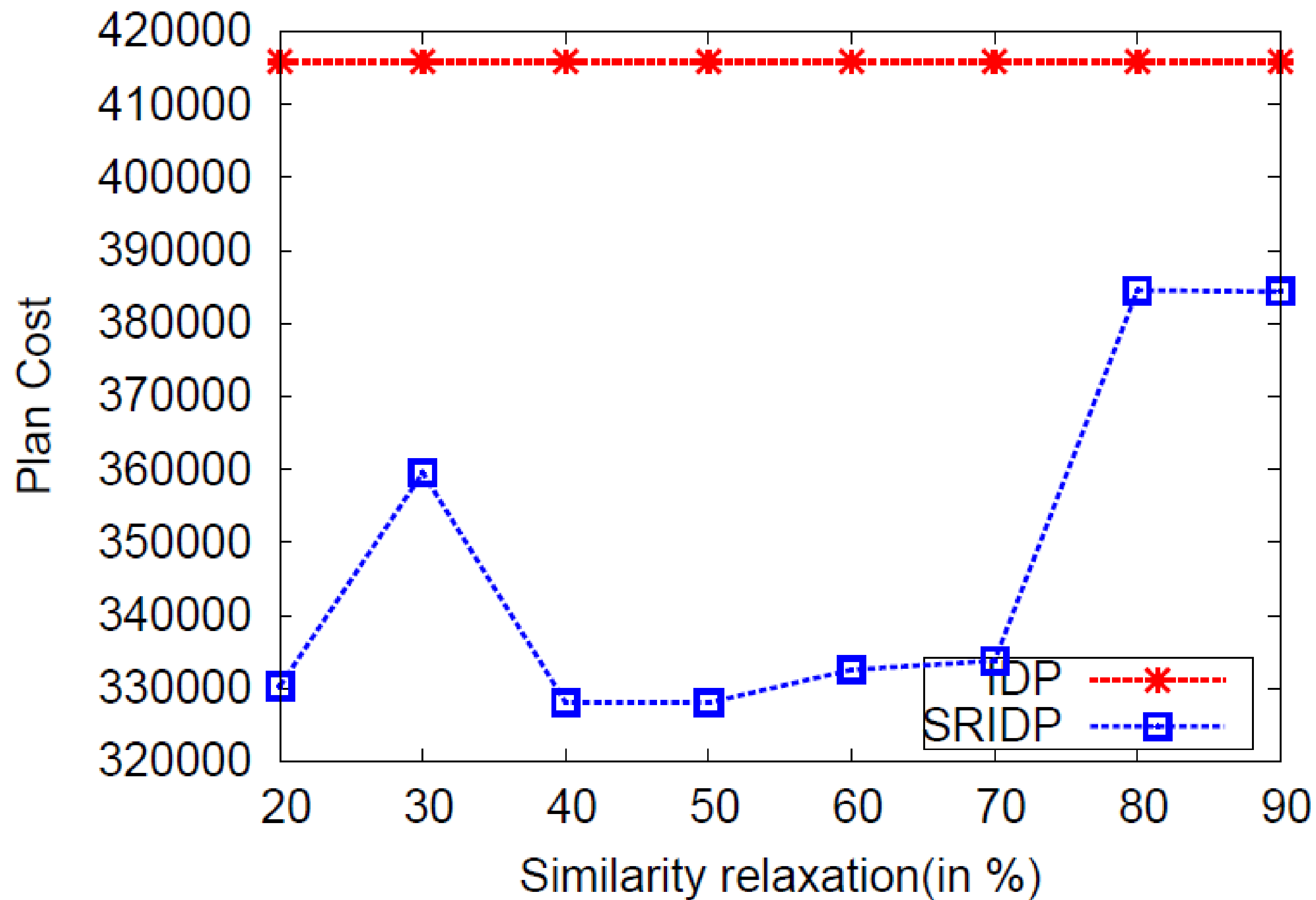


Varying density level and similarity error bound

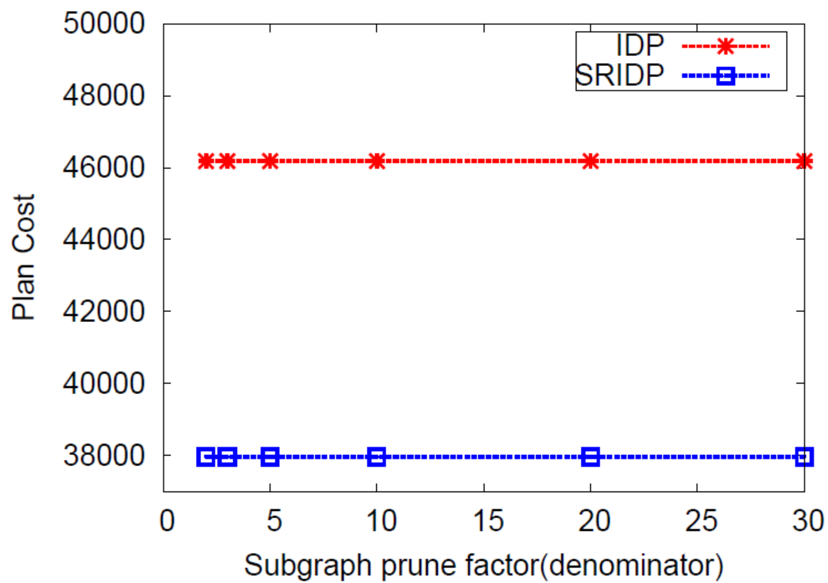


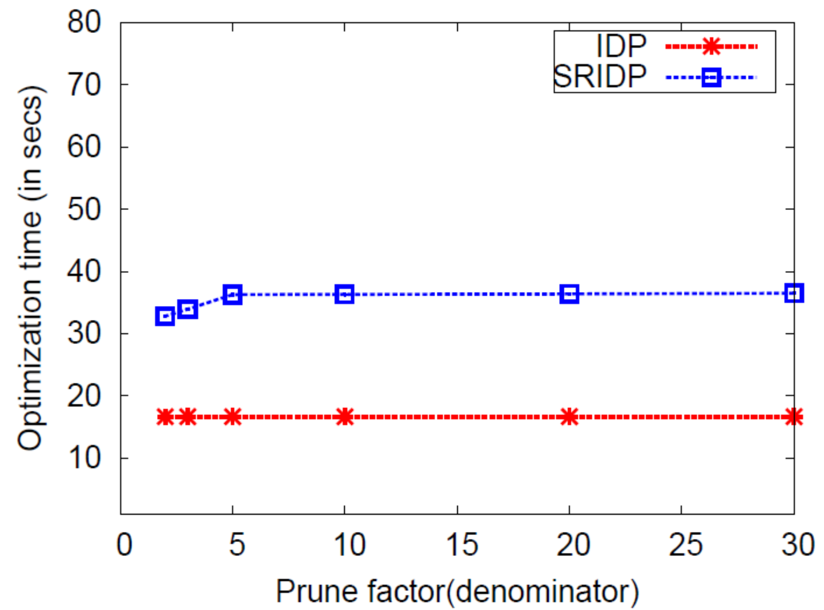


Varying similarity bounds for an 18 table query



Varying subgraph set prune factor





- We proposed a memory efficient similar subgraph reuse scheme to stretch DP based algorithms to crawl up the lattice
- However our scheme suited better for IDP
- Stretching the performance without compromising optimality
- No join candidates were pruned
- SRIDP generates better plans than IDP

Future Work

- Extending SRIDP to modern hardware by multi-threading
  - This could ask for efficient distribution of join candidates across various threads
  - Mainly locality improvement so that re-usable join candidates are grouped together
  - Load balancing

References

[1] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. on Database Systems,* 25:2000, 1998.

[2] Qiang Zhu, Yingying Tao, and Calisto Zuzarte. Optimizing complex queries based on similarities of subqueries. *Knowl. Inf. Syst., 8(3):350–373, 2005.*

[3] Gopal Chandra Das and Jayant R. Haritsa. Robust heuristics for scalable optimization of complex sql queries. In *ICDE, pages 1281–1283, 2007.*

References

[1] Finding communities by clustering a graph into overlapping subgraphs.

[2] Yu Wang 0014 and Carsten Maple. A novel efficient algorithm for determining maximum common subgraphs. In *IV, pages 657–663, 2005.*

[3] Ivan T. Bowman and G. N. Paulley. Join enumeration in a memory-constrained environment. In *ICDE, pages 645–654, 2000.*

[4] Gopal Chandra Das and Jayant R. Haritsa. Robust heuristics for scalable optimization of complex sql queries. In *ICDE, pages 1281–1283, 2007.*

[5] David DeHaan and Frank Wm. Tompa. Optimal top-down join enumeration. In *SIGMOD Conference, pages 785–796, 2007.*

[6] C´esar A. Galindo-Legaria and Milind Joshi. Orthogonal optimization of subqueries and aggregation. In *SIGMOD Conference, pages 571–581, 2001.*

[7] C´esar A. Galindo-Legaria, Arjan Pellenkoft, and Martin L. Kersten. Fast, randomized join-order selection - why use transformations? In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, pages 85–95. Morgan Kaufmann, 1994.*

References (cont'd)

[8] C´esar A. Galindo-Legaria and Arnon Rosenthal. How to extend a conventional optimizer to handle one- and two-sided outerjoin. In *ICDE, pages 402–409,* 1992.

[9] Arianna Gallo, Pauli Miettinen, and Heikki Mannila. Finding subgroups having several descriptions: Algorithms for redescription mining.

[10] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv., 25(2):73–170, 1993.*

[11] Jorng-Tzong Horng, Baw-Jhiune Liu, and Cheng-Yan Kao. A genetic algorithm for database query optimization. In *International Conference on Evolutionary Computation, pages 350–355, 1994.*

[12] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *SIGMOD Conference, pages 168–177, 1991.*

[13] Yannis E. Ioannidis and Eugene Wong. Query optimization by simulated annealing. In *SIGMOD Conference, pages 9–22, 1987.*

[14] Jean jacques Hbrard. A direct algorithm to find a largest common connected induced subgraph of two graphs.

References (cont'd)

[15] Matthias Jarke and J¨urgen Koch. Query optimization in database systems. *ACM Comput. Surv., 16(2):111–152, 1984.*

[16] H. Jiang and C. W. Ngo. Image mining using inexact maximal common subgraph of multiple args. In *Int. Conf. on Visual Information Systems, 2003.*

[17] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. on Database Systems,* 25:2000, 1998.

[18] Rosana S. G. Lanzelotte, Patrick Valduriez, and Mohamed Za¨ıt. On the effectiveness of optimization search strategies for parallel execution spaces. In *VLDB, pages 493–504, 1993.*

[19] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Softw., Pract. Exper., 12(1):23–34, 1982.*

[20] Guido Moerkotte. Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In *In Proc. 32nd International Conference on Very Large Data Bases, pages* 930–941, 2006.

References(cont'd)

[21] Guido Moerkotte. Dp-counter analytics. Technical report, 2006.

[22] Guido Moerkotte and Thomas Neumann. Dynamic programming strikes back. In *SIGMOD Conference, pages 539–552, 2008.*

[23] Tadeusz Morzy, Maciej Matysiak, and Silvio Salza. Tabu search optimization of large join queries. In *EDBT, pages 309–322, 1994.*

[24] Kiyoshi Ono and Guy M. Lohman. Measuring the complexity of join enumeration in query optimization. In Dennis McLeod, Ron Sacks-Davis, and Hans-J¨org Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings, pages 314–*325. Morgan Kaufmann, 1990.

[25] Jun Rao, Bruce G. Lindsay, Guy M. Lohman, Hamid Pirahesh, and David E. Simmen. Using eels, a practical approach to outerjoin and antijoin reordering. In *ICDE, pages 585–594, 2001.*

[26] Matthias Rarey and J. Scott Dixon. Feature trees: A new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design, 12(5):471–490, 1998.*

References (cont'd)

[27] John W. Raymond and Peter Willett 0002. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design, 16(7):521–533, 2002.*

[28] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal, 45:2002, 2002.*

[29] Patricia G. Selinger and Michel E. Adiba. Access path selection in distributed database management systems. In *ICOD, pages 204–215, 1980.*

[30] Timos K. Sellis. Global query optimization. In *SIGMOD Conference, pages* 191–205, 1986.

[31] Leonard D. Shapiro, David Maier, Paul Benninghoff, Keith Billings, Yubo Fan, Kavita Hatwal, Quan Wang, Yu Zhang, Hsiao min Wu, and Bennet Vance. Exploiting upper and lower bounds in top-down query optimization. In *IDEAS, pages 20–33, 2001.*

[32] Arun N. Swami. Optimization of large join queries: Combining heuristic and combinatorial techniques. In *SIGMOD Conference, pages 367–376, 1989.*

[33] Arun N. Swami and Balakrishna R. Iyer. A polynomial time algorithm for optimizing join queries. In *ICDE, pages 345–354, 1993.*

[34] Akutsu Tatsuya. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE transactions on fundamentals of electronics, communications and computer sciences, 76(9):1488–1493,* 1993-09-25.

[35] Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM,* 23(1):31–42, 1976.

[36] Bennet Vance and David Maier. Rapid bushy join-order optimization with cartesian products. In *In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 35–46, 1996.*

References (cont'd)

[37] P. Viswanath, M. Narasimha Murty, and Shalabh Bhatnagar. Fusion of multiple
approximate nearest neighbor classifiers for fast and efficient classification.
Information Fusion, 5(4):239–250, 2004.

[38] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns.
In *KDD, pages 286–295, 2003.*

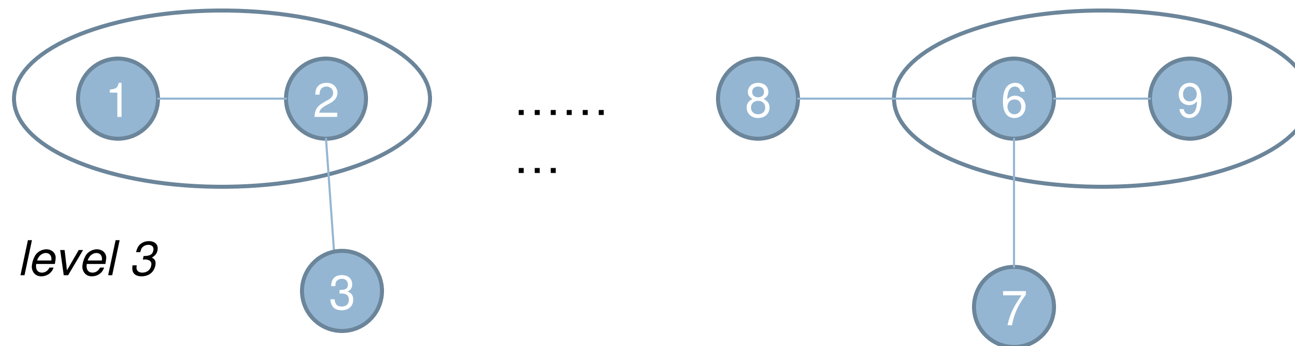
[39] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in
graph databases. In *SIGMOD Conference, pages 766–777, 2005.*

[40] Qiang Zhu, Yingying Tao, and Calisto Zuzarte. Optimizing complex queries
based on similarities of subqueries. *Knowl. Inf. Syst., 8(3):350–373, 2005.*

END

Pruning to reduce DP search space (cont'd)

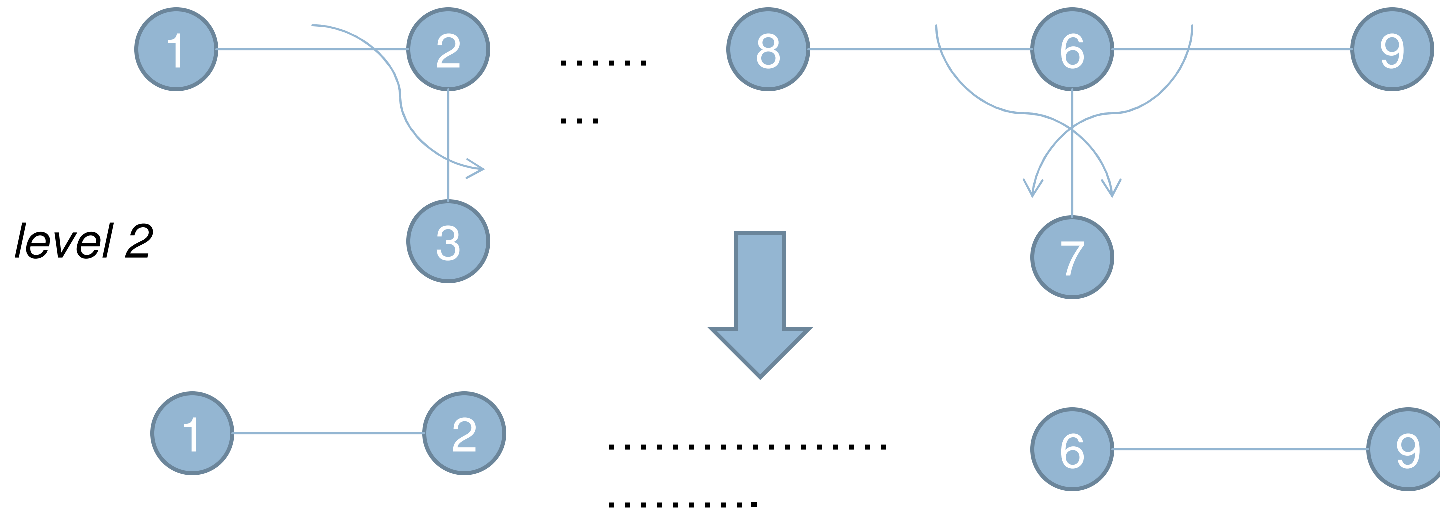
- New hubs



- (1,2,5) (5 isn't seen) may survive , (6,8,9) may survive
- Possibly , for a join candidate (6,7,9, 10) arriving at a later point, (6,7,9) with 10 may be the ideal combination but that wouldn't be available anymore
- Pruning join candidates is an impediment to plan quality
- We aim at total avoidance of pruning join candidates

Pruning to reduce DP search space

- Identify hubs (nodes collectively or individually having a high degree) at each level in the query graph.
- Apply pruning on join candidates using a skyline on (cost, cardinality, selectivity)



- IDP has polynomial time and space complexity of the order of $O(n^k)$.
- In this analysis, k (the size of the building blocks) is considered to be constant, and n (the number of tables) are the variables which depend on the query to optimize.