Monitoring Aggregate k-NN Objects in Road Networks

Lu Qin, Jeffrey Xu Yu, Bolin Ding , Yoshiharu Ishikawa

Outline

- Introduction
- Problem Definition
- Existing Solution
- A Novel Approach
- Experimental Studies
- Summary

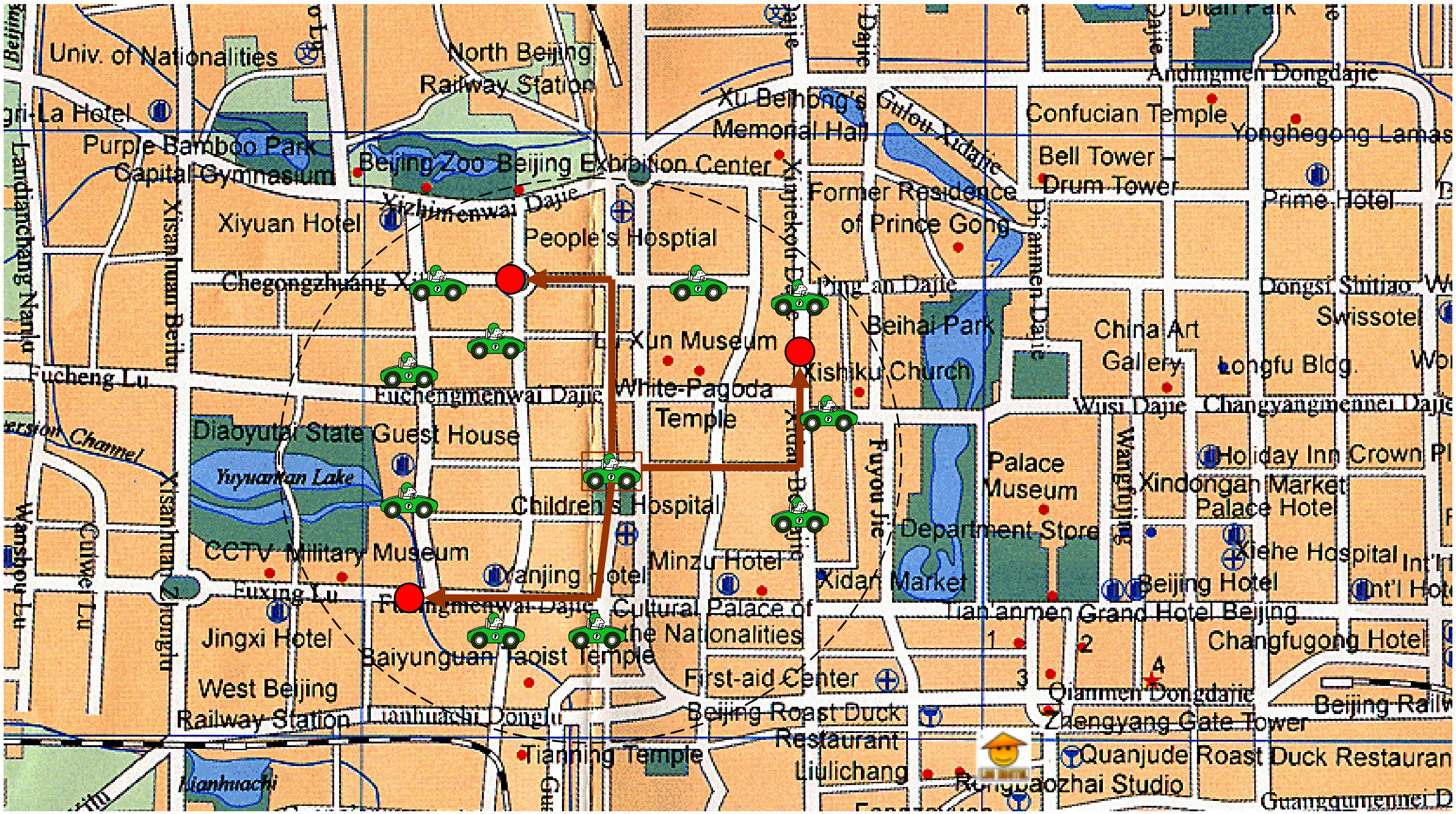
Introduction

Emergency Center

Nearest Path

Moving Car

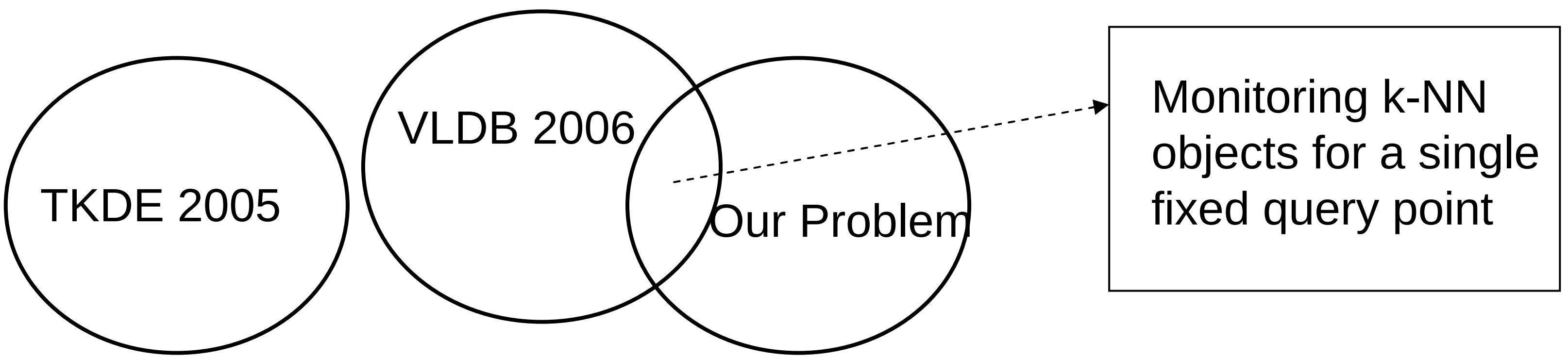
Car in Danger



Introduction

Our Problem: Monitoring k-NN objects over a road network to minimize (or maximize) an aggregate distance function for multiple query points.

	VLDB 2006	TKDE 2005	Our Problem
Continuous Query?	Yes	No	Yes
Object Points Can Move?	Yes	No	Yes
Query Points Can Move?	Yes	No	No
Multiple Query Points?	No	Yes	Yes

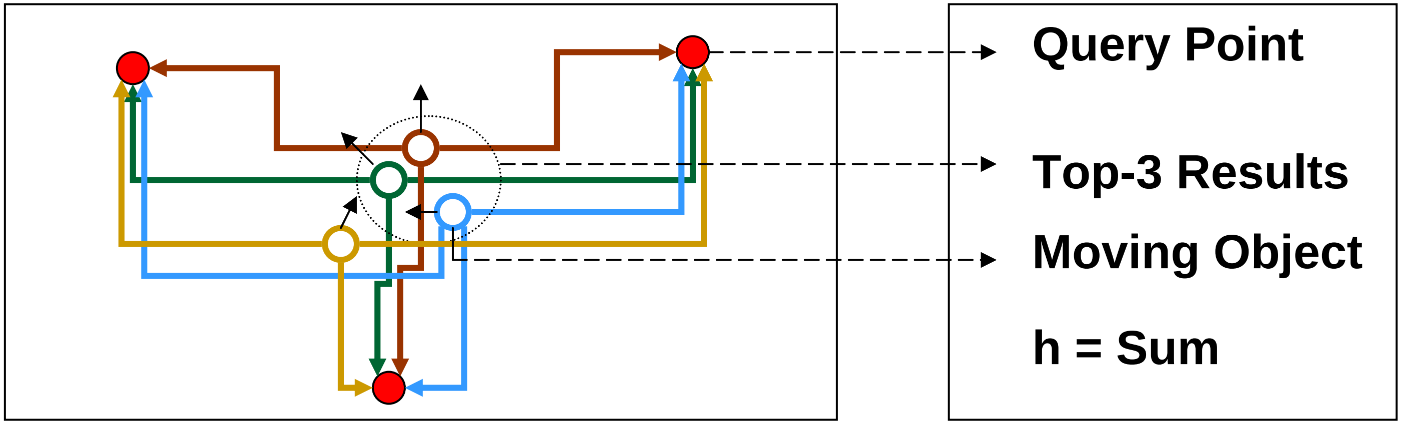


Problem Definition

Continuous Nearest Neighbor Query CANN(Q,k,h):

- Q: a set of fixed query points over a road network.
- k: a positive integer.
- h: an aggregate function (e.g. Sum, Min, Max).
  - Defined on points in Q
  - Variables are moving objects
  - Network distance
- Result: monitoring the top-k moving objects that has the smallest h function values in the road network.

Timestamp 1

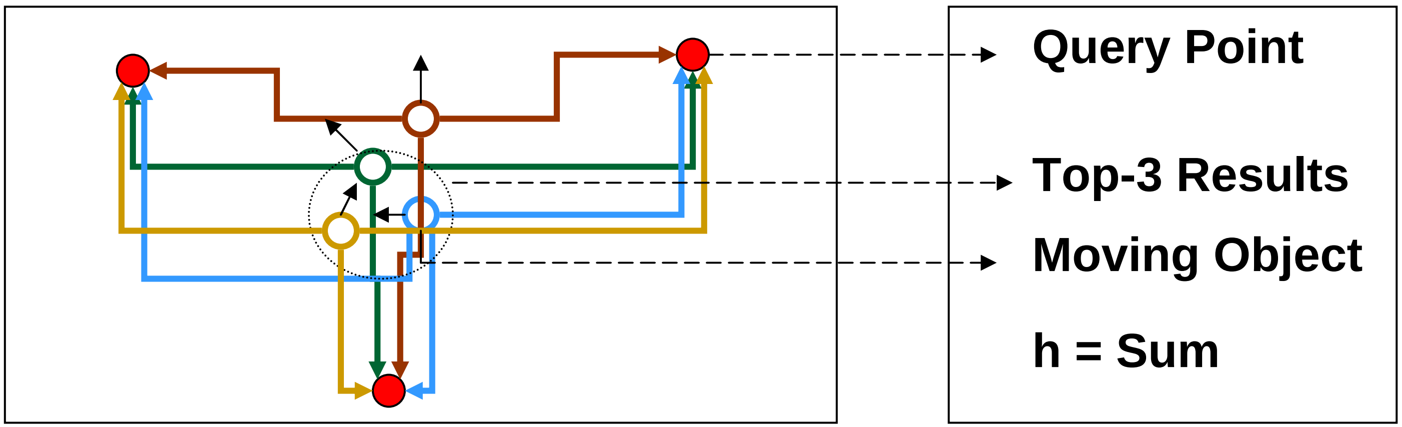


Problem Definition

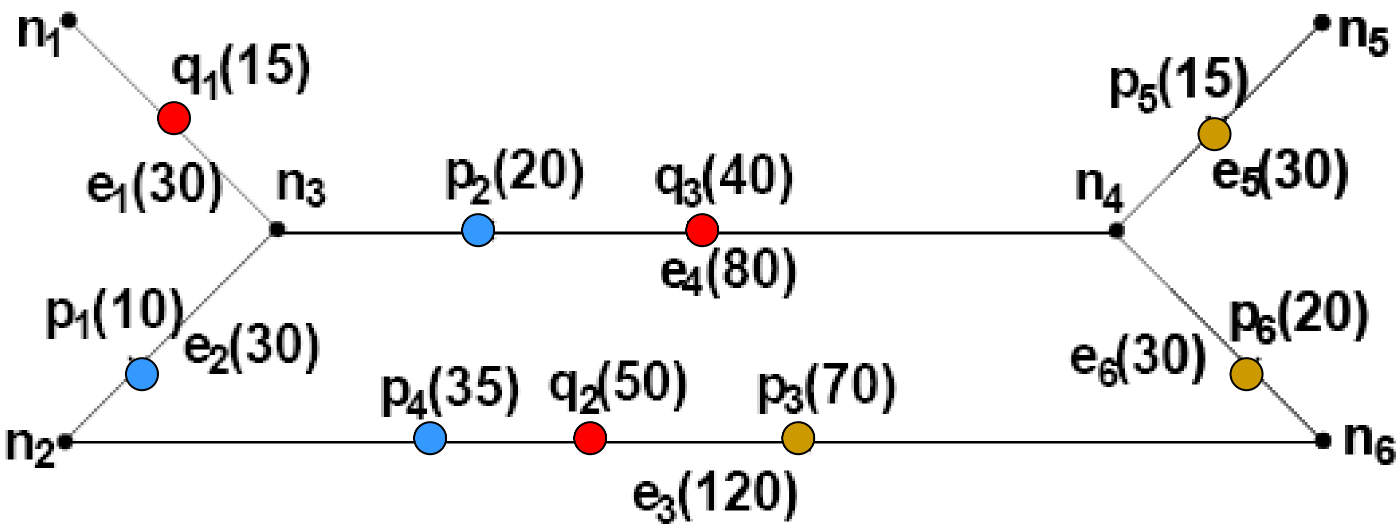
Continuous Nearest Neighbor Query CANN(Q,k,h):

- Q: a set of fixed query points over a road network.
- k: a positive integer.
- h: an aggregate function (e.g. Sum, Min, Max).
  - Defined on points in Q
  - Variables are moving objects
  - Network distance
- Result: monitoring the top-k moving objects that has the smallest h function values in the road network.

Timestamp 2

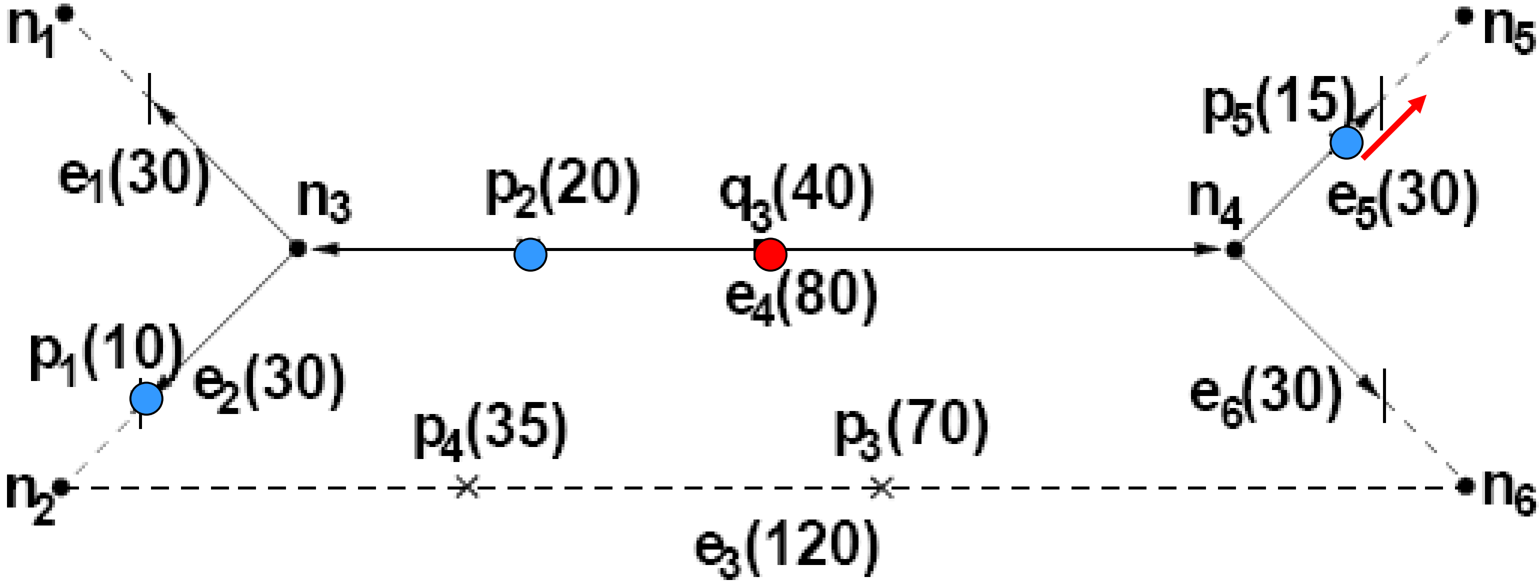


Problem Definition



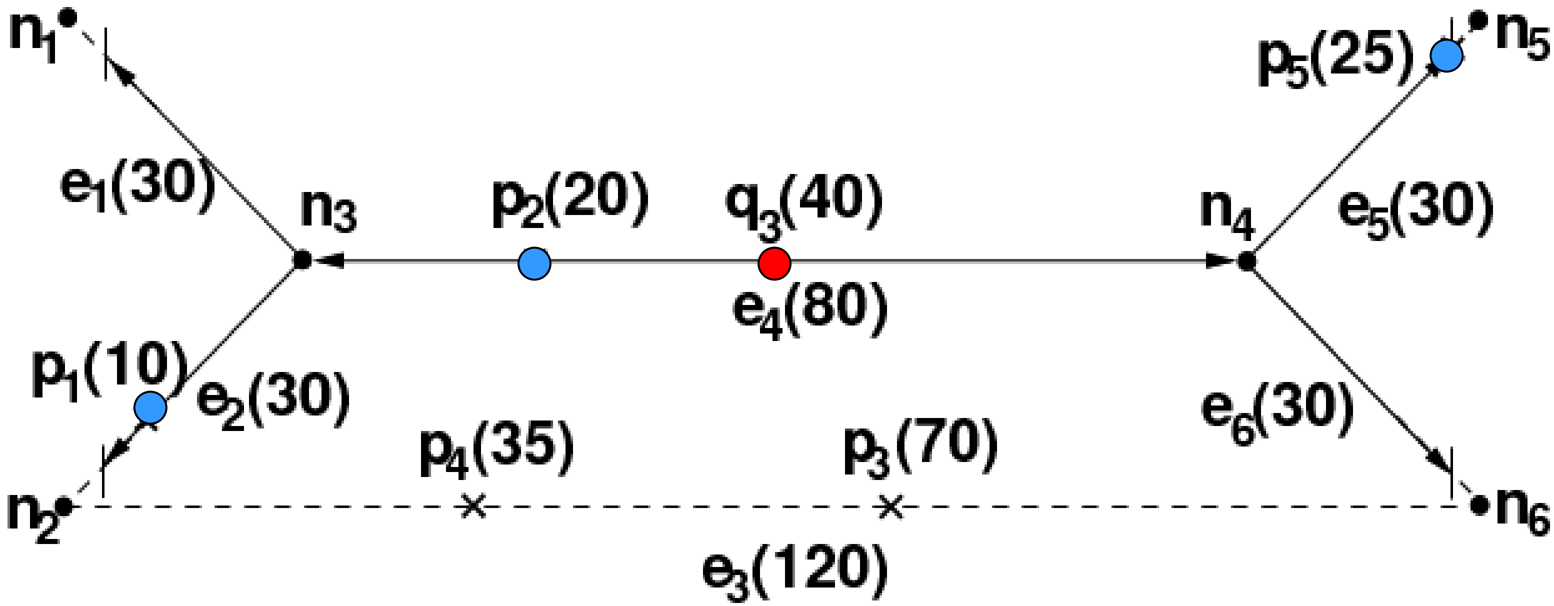
- A Running Example
  - Q={q1,q2,q3}, k=3, and h=Sum.
  - Sum(p1)=Sum{d(p1,q1), d(p1,q2), d(p1,q3)}=155, Sum(p2)=155, Sum(p3)=255, Sum(p4)=200, Sum(p5)=288, and Sum(p6)=255.
  - The top-3 result is {p1, p2, p4}.

VLDB06: Tree-Expand-Approach



- A tree expand approach for a single query point.
- Expanding and Shrinking
- Timestamp 1: CANN({q3},3,sum)={p2,p5,p1}

VLDB06: Tree-Expand-Approach

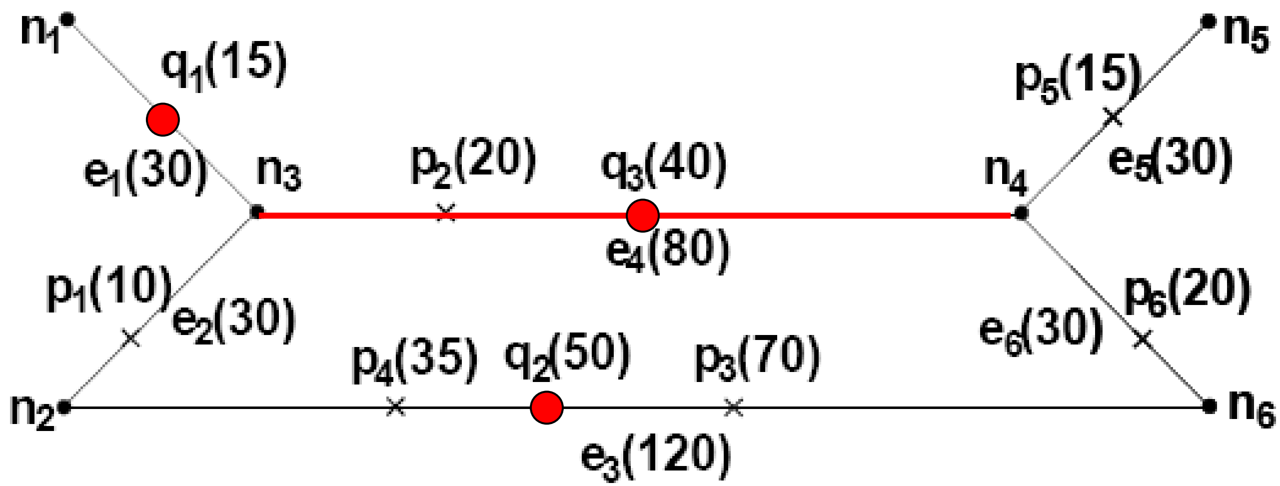


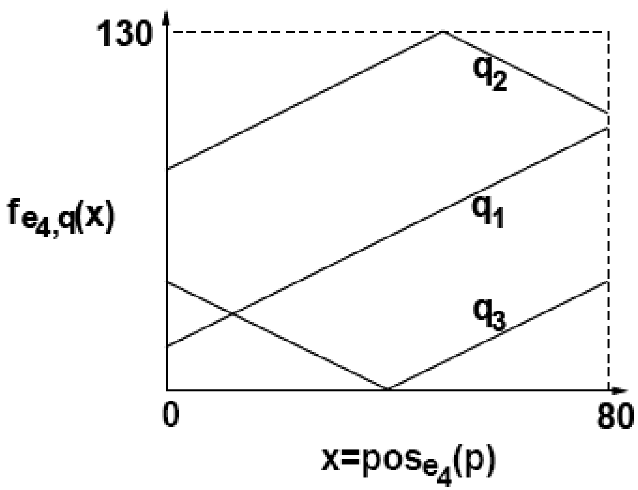
- A tree expand approach for a single query point.
- Expanding and Shrinking
- Timestamp 2: CANN({q3},3,sum)={p2,p1,p5}

Our Solution: Non-Tree-Expand

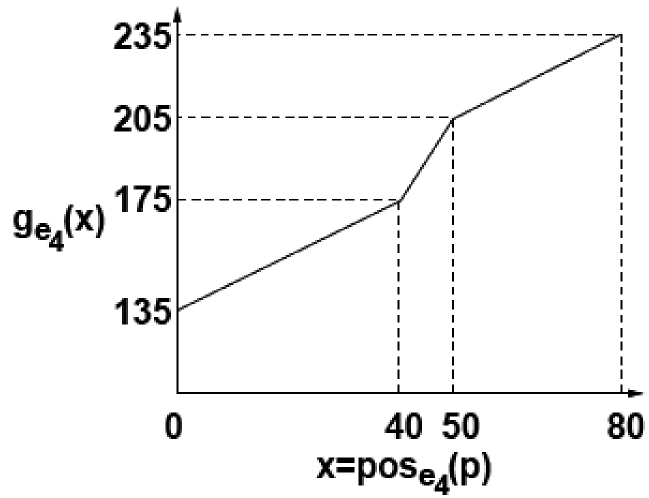
- The order of visiting edges – Not a Tree
- A two-step approach
  - Step 1: Construct a Query Graph (External Structure)
    - Study the aggregate functions on edges.
    - Pre-compute as much information as possible.
    - Find an order of visiting edges.
  - Step 2: Monitoring Top-k Objects
    - Sequential Access
    - Initial result computation
    - Avoid re-computation

Query Graph: Functions on Edges



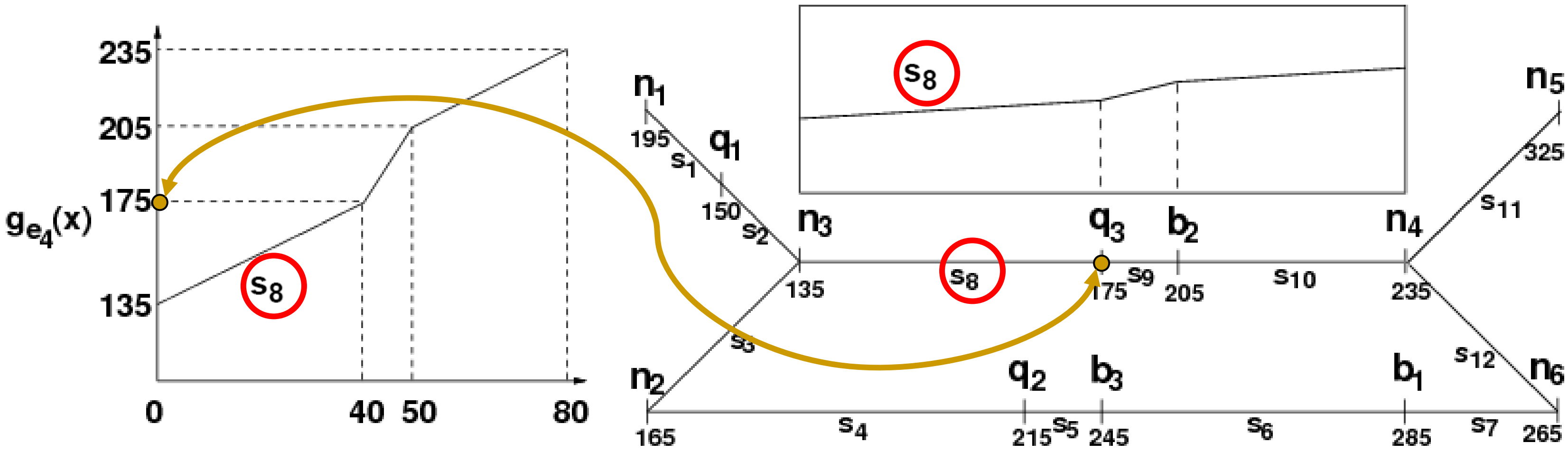


(a) 1-source edge functions



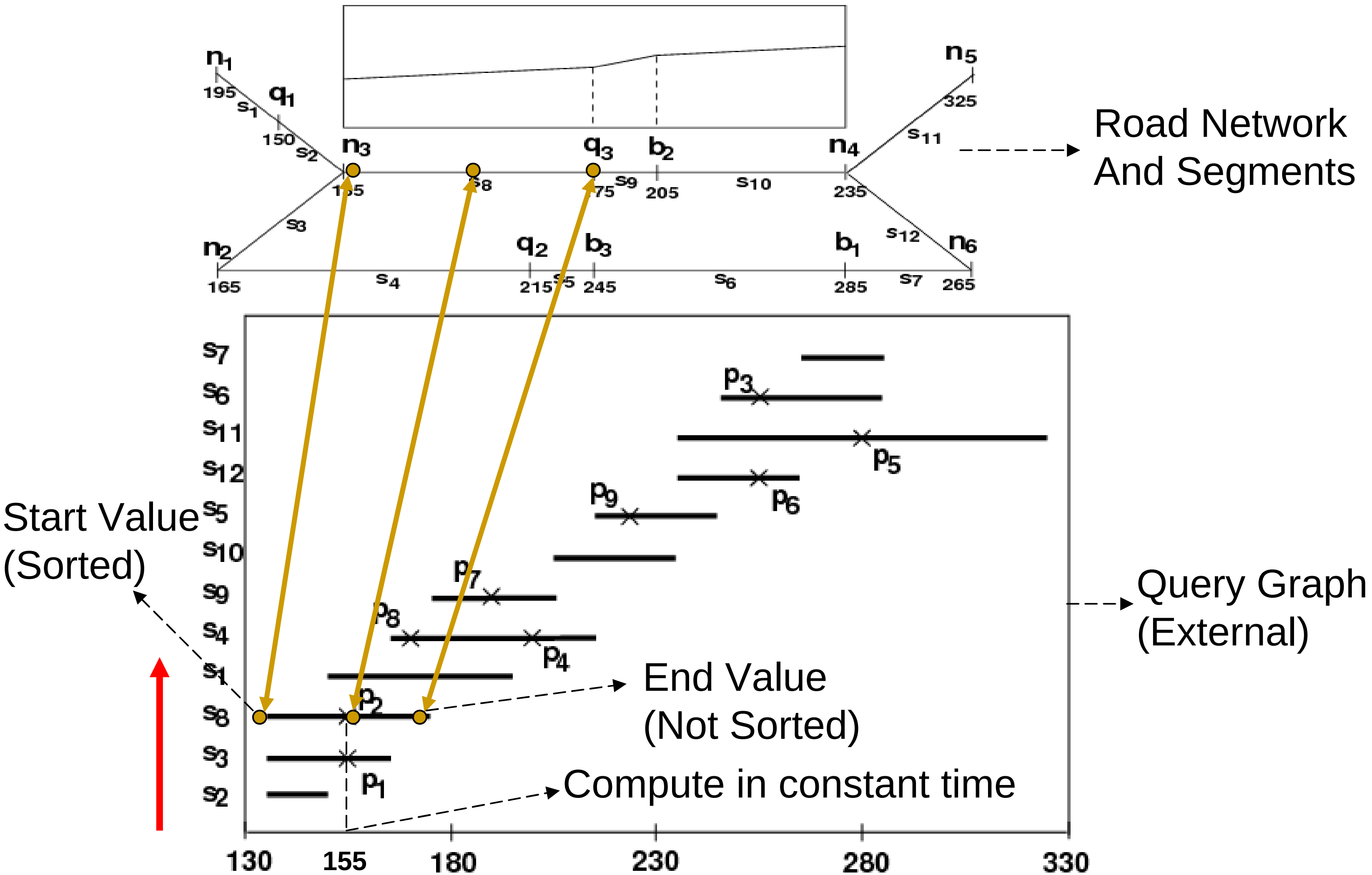
(b) Sum

Query Graph: Segmentation



- Segmentation based on the aggregate piecewise linear function.
- s_8 Start Value: 135; End Value: 175.

Query Graph : Sorting Segments

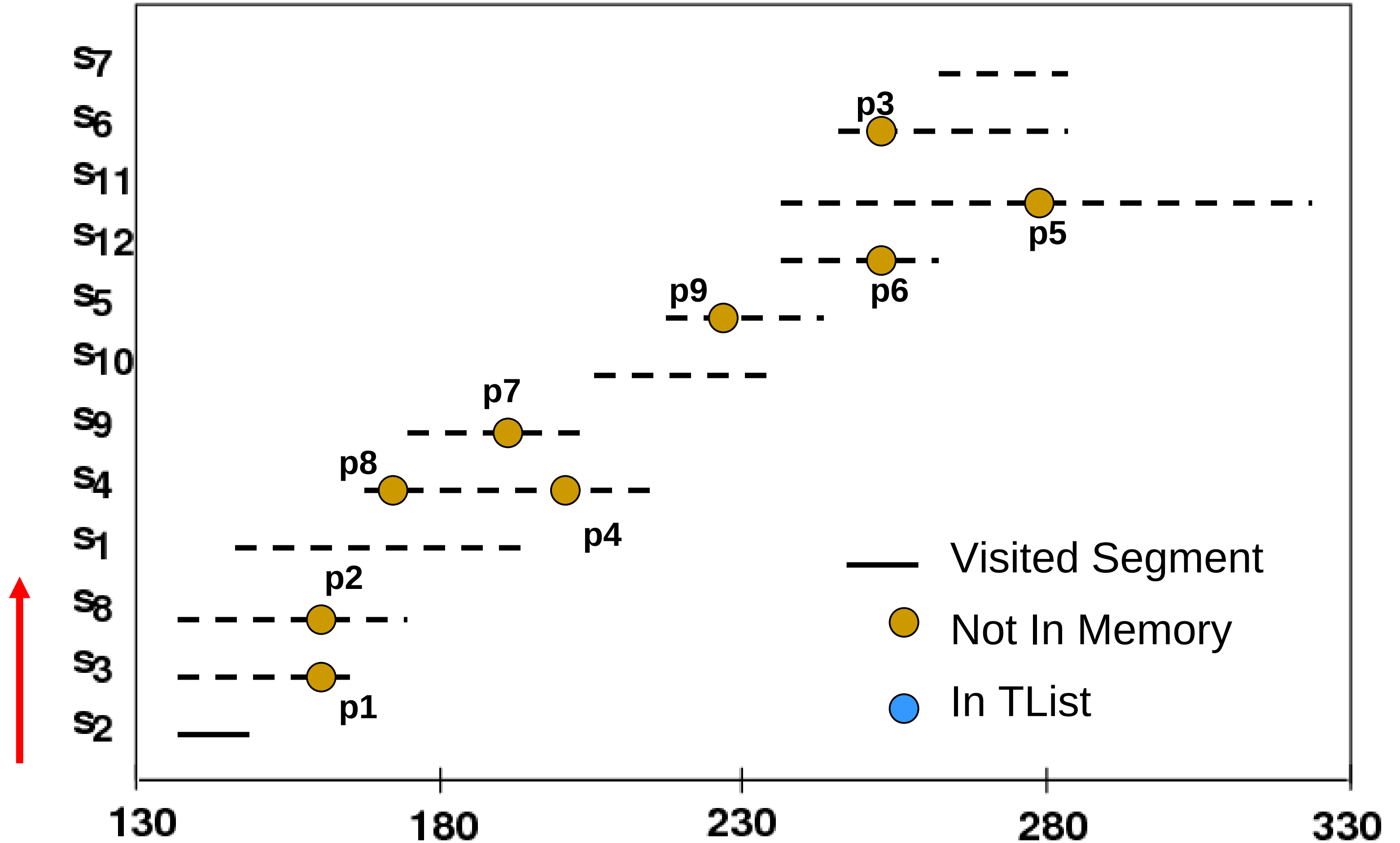


Monitoring Algorithm: Initial Result

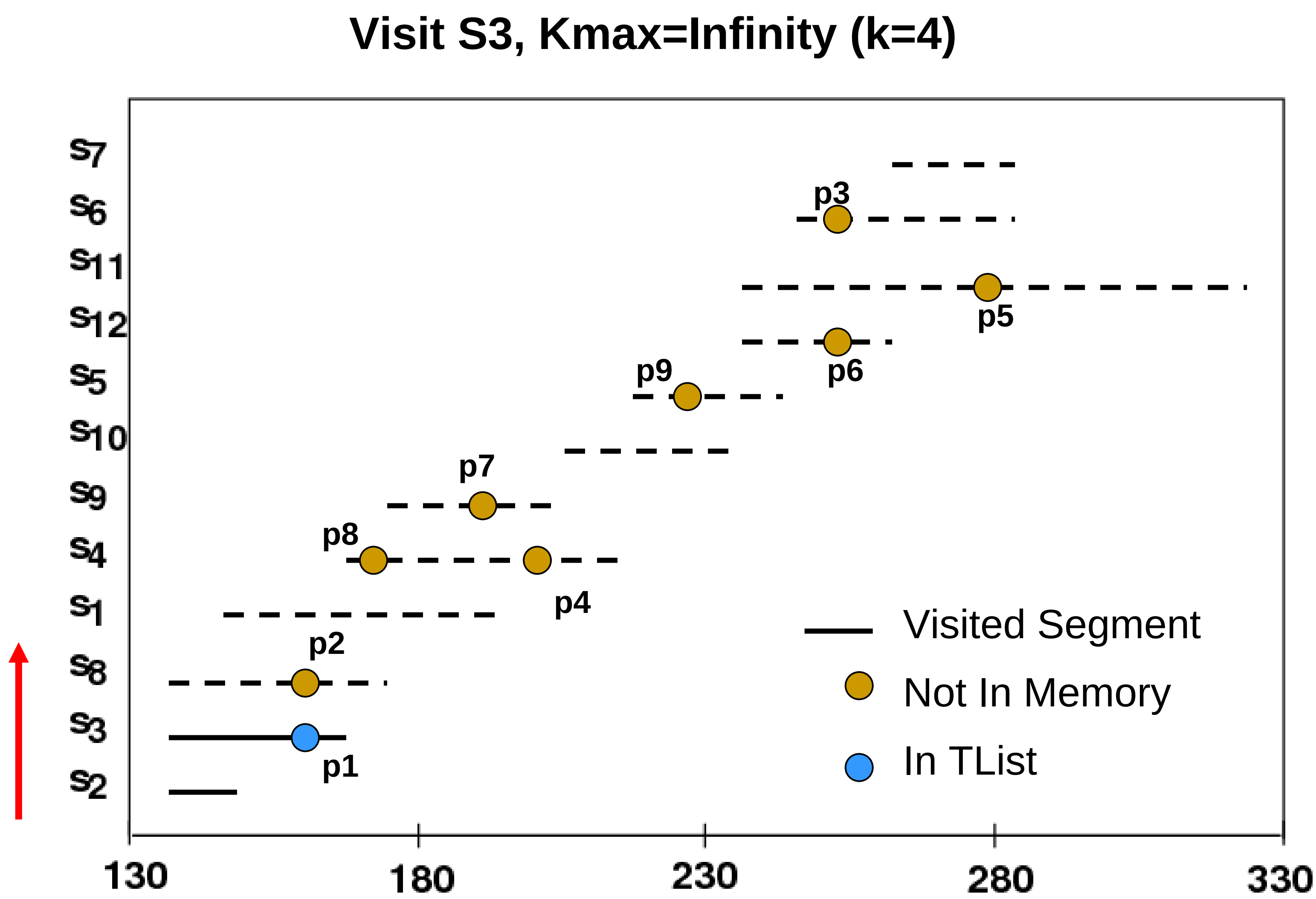
- **TList: The current top-k objects in ascending order of their h values**
- **Tmax: the h value of the k-th object in TList**
- **Sequential access**
- **Stop condition: Tmax <= Start value of the next segment**

Initial Result Computation

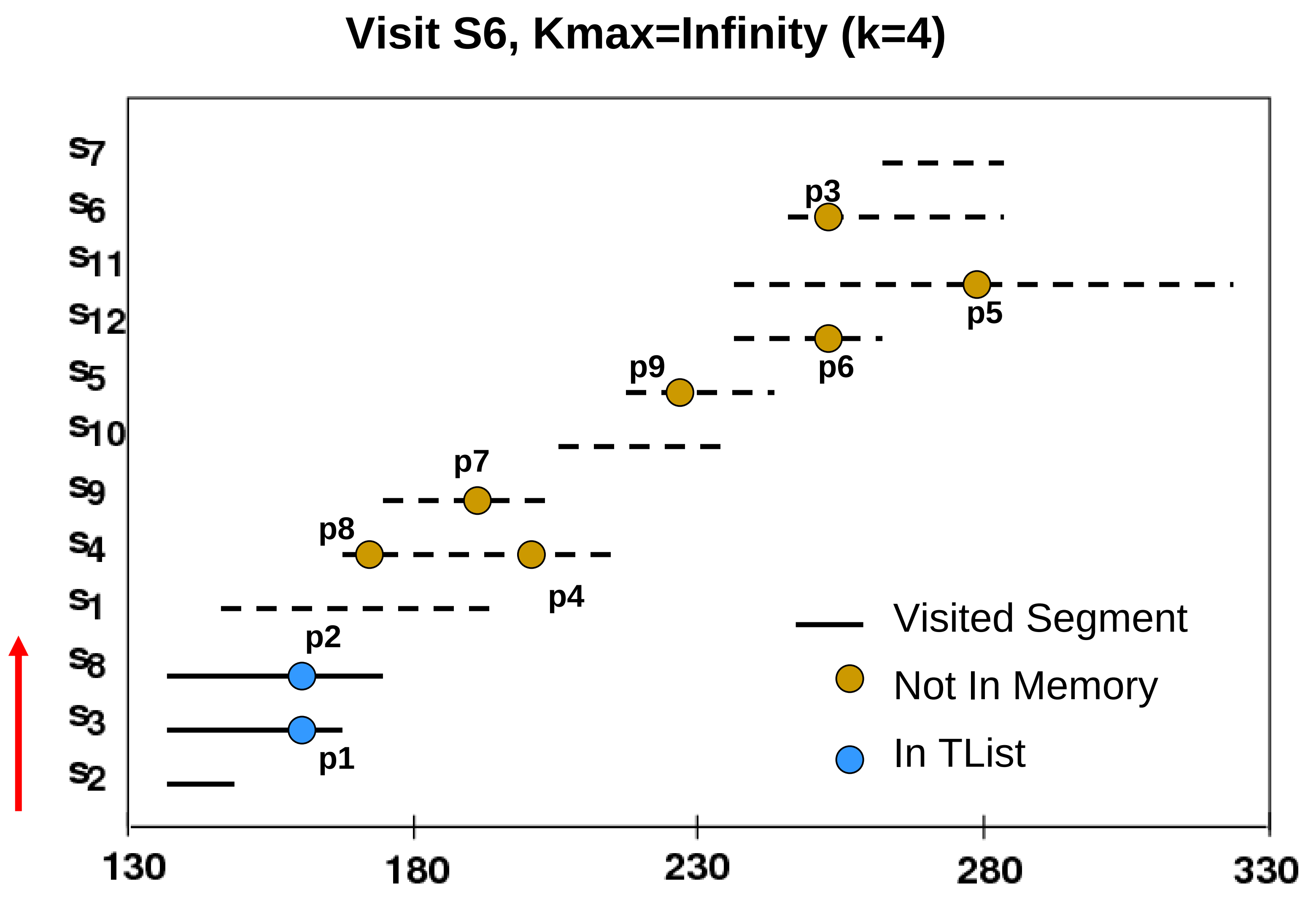
Visit S2, Kmax=Infinity (k=4)



Initial Result Computation

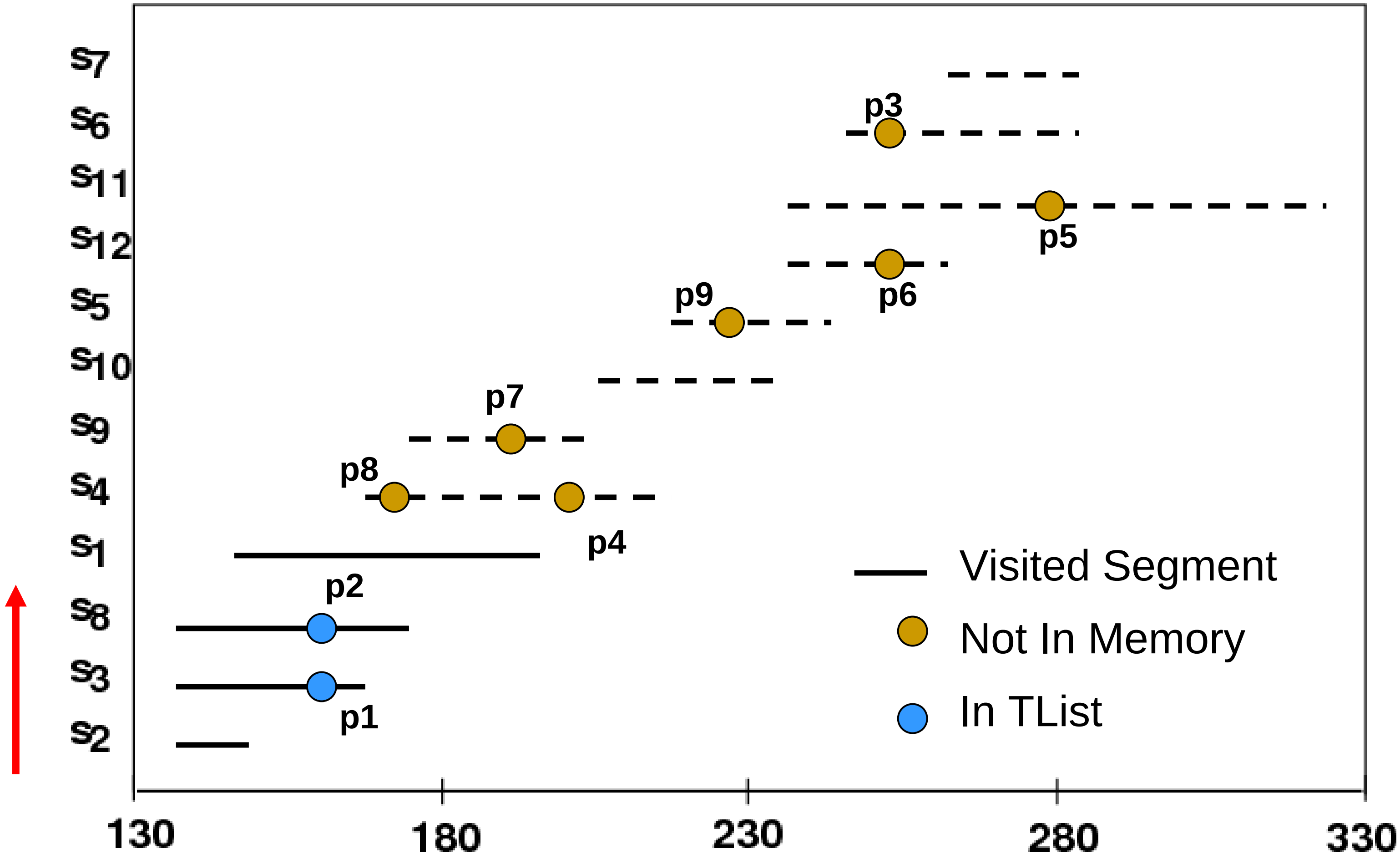


Initial Result Computation



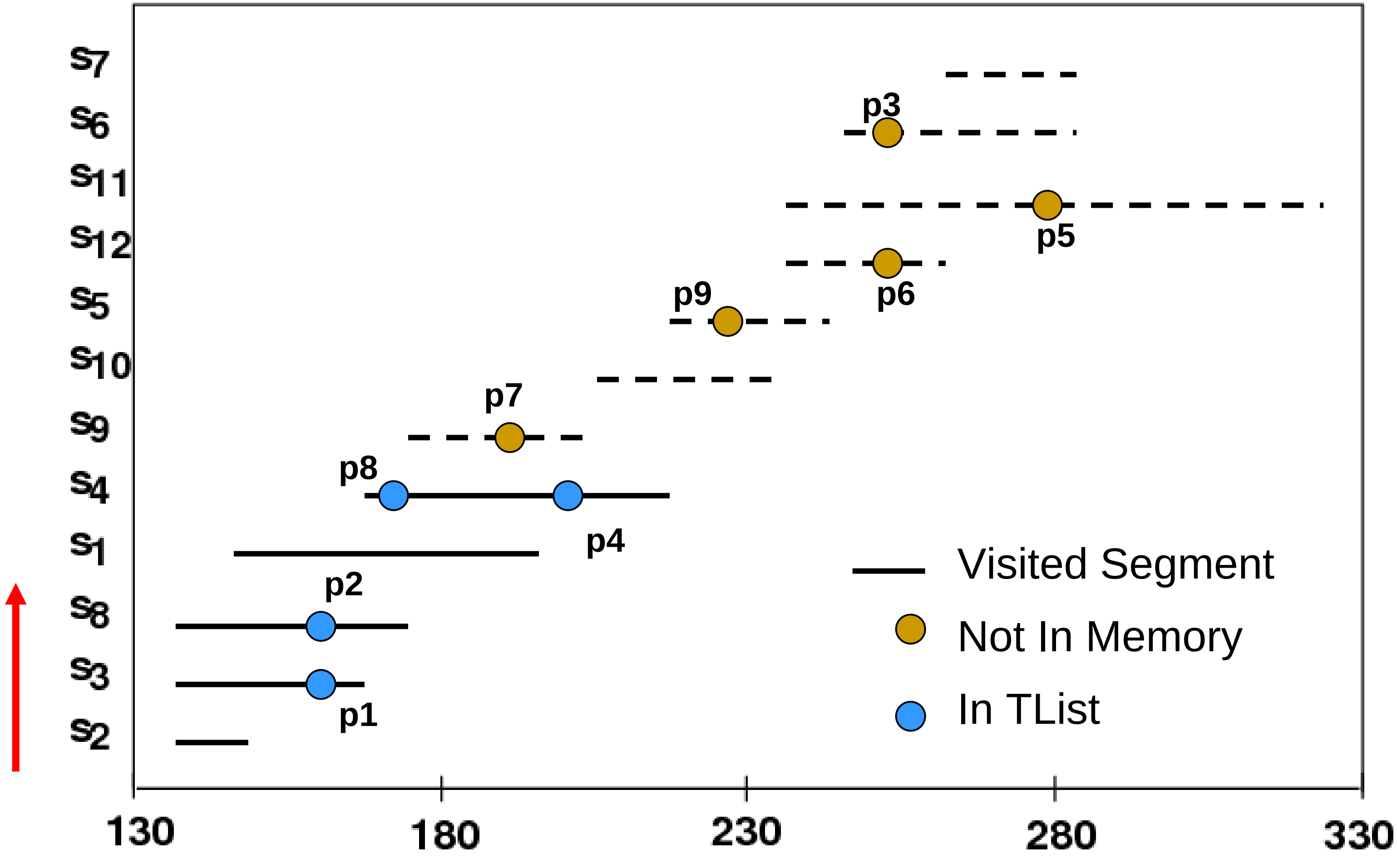
Initial Result Computation

Visit S1, Kmax=Infinity (k=4)



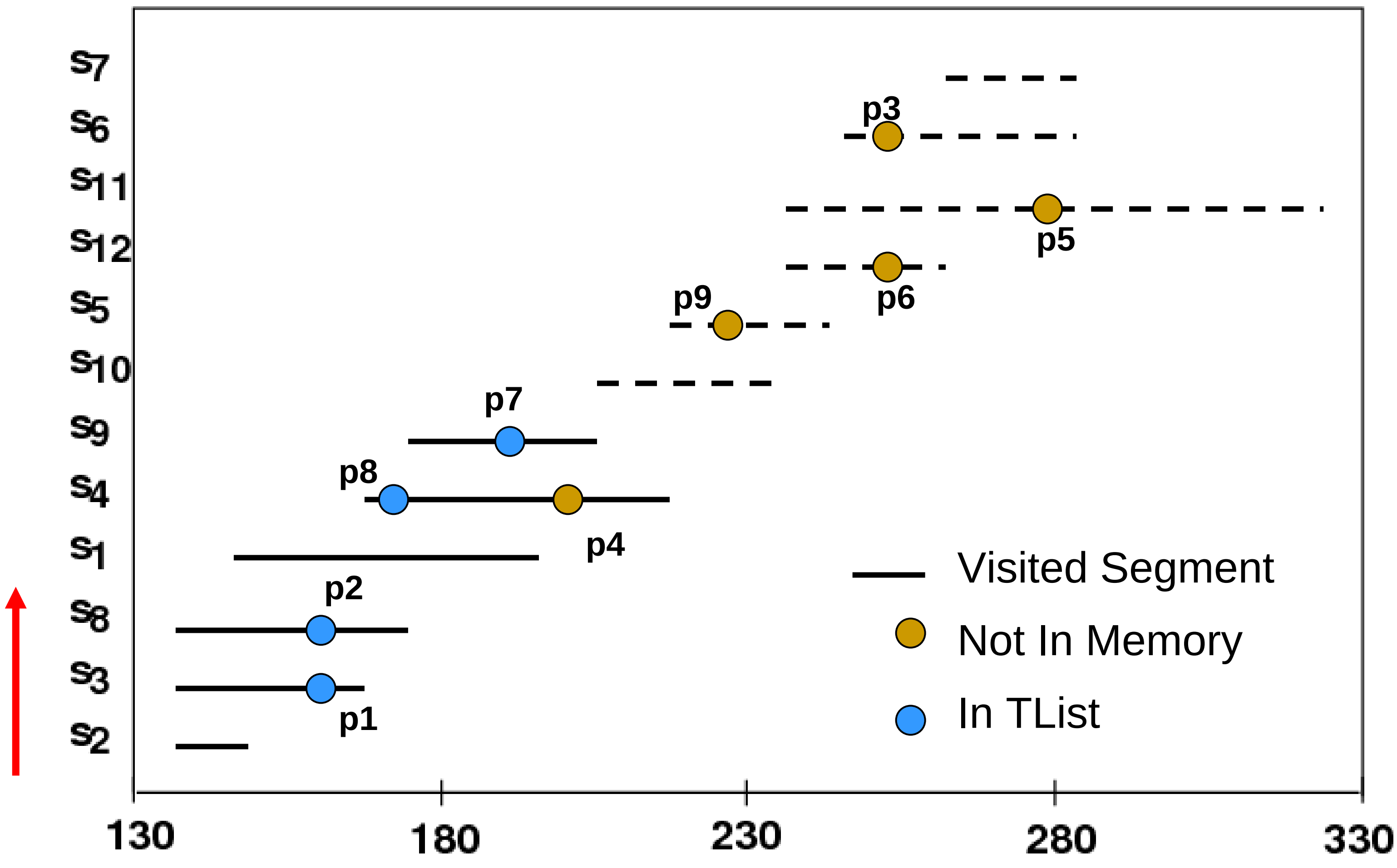
Initial Result Computation

Visit S4, Kmax=200 (k=4)



Initial Result Computation

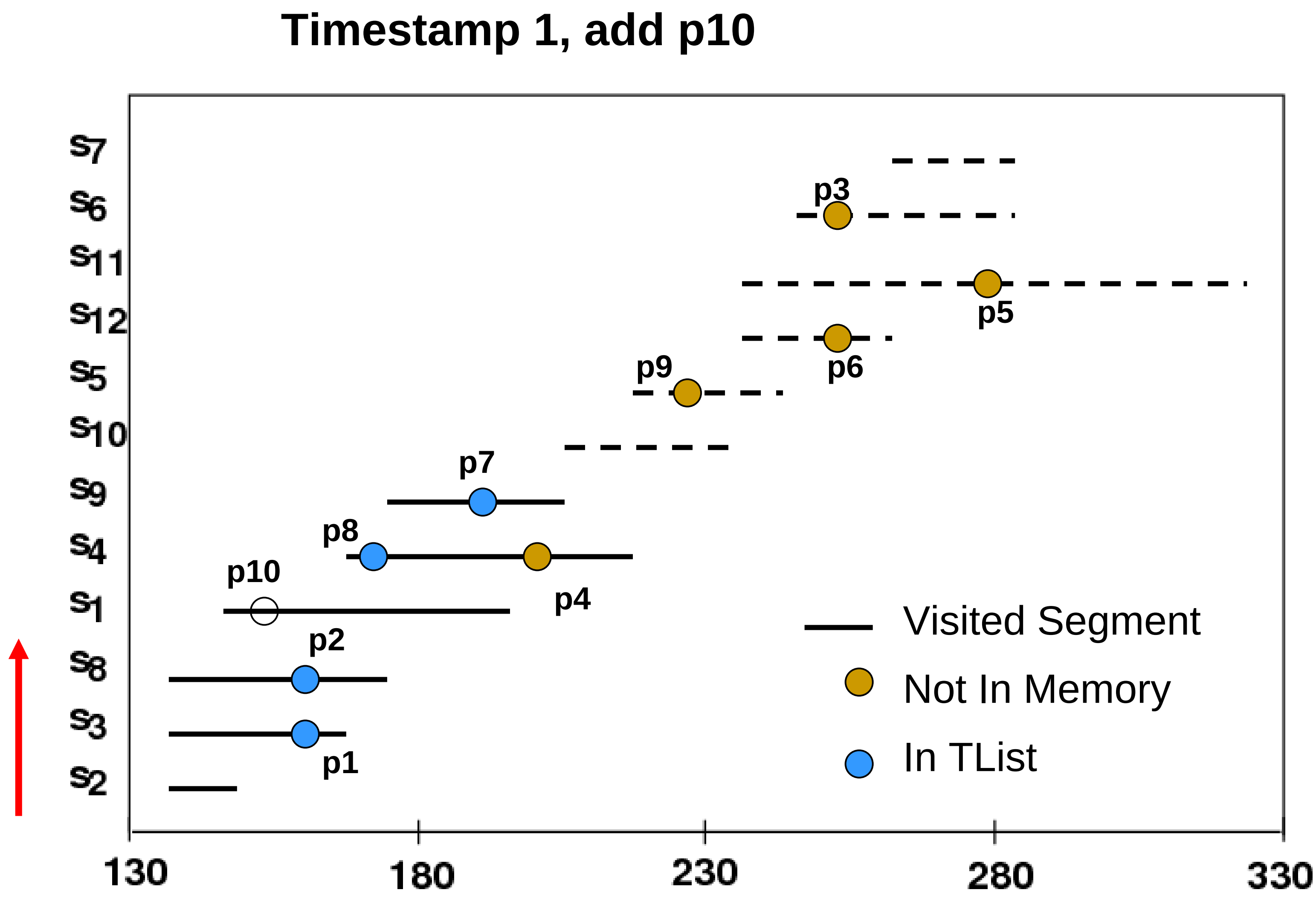
Visit S9, Kmax=190 (k=4), Stop, Report TList



Naïve Monitoring Algorithm

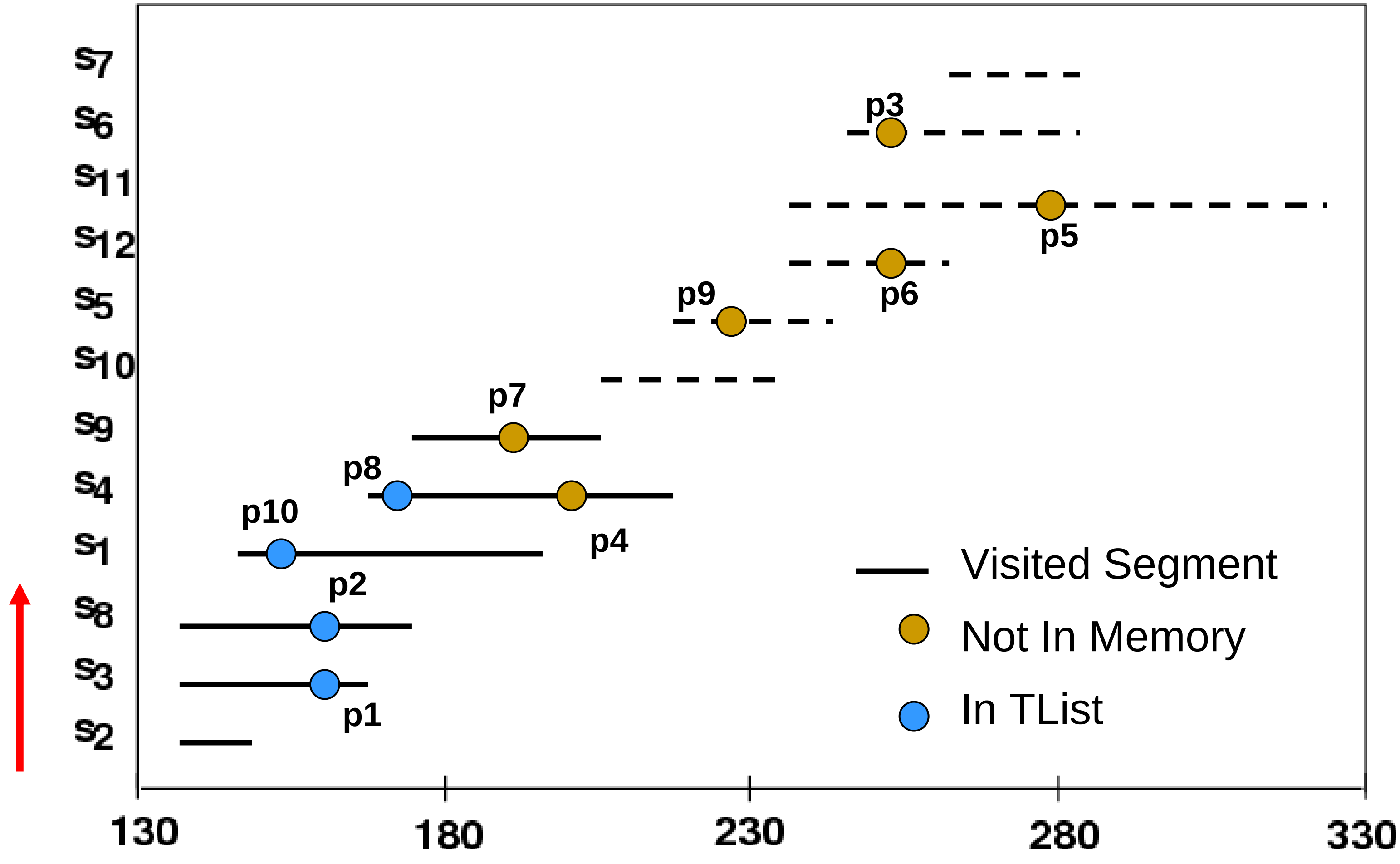
- **Situation 1, Add an Object to TList: Incremental Update**
- **Situation 2, Remove an Object in TList: Re-computation**
- **Situation 3, Otherwise: Do nothing**

Naïve Monitoring Algorithm



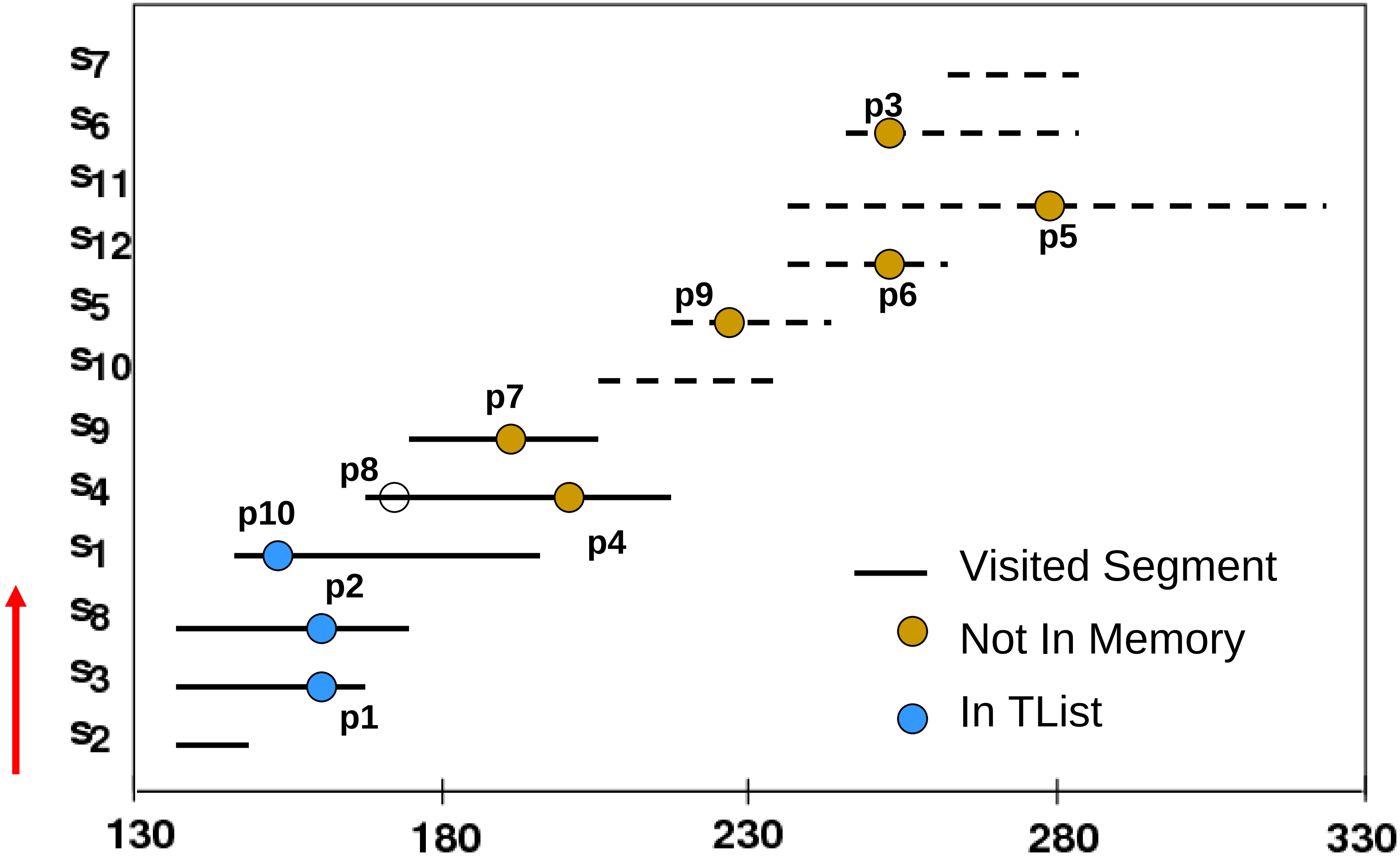
Naïve Monitoring Algorithm

Timestamp 1, add p10: Incremental Update



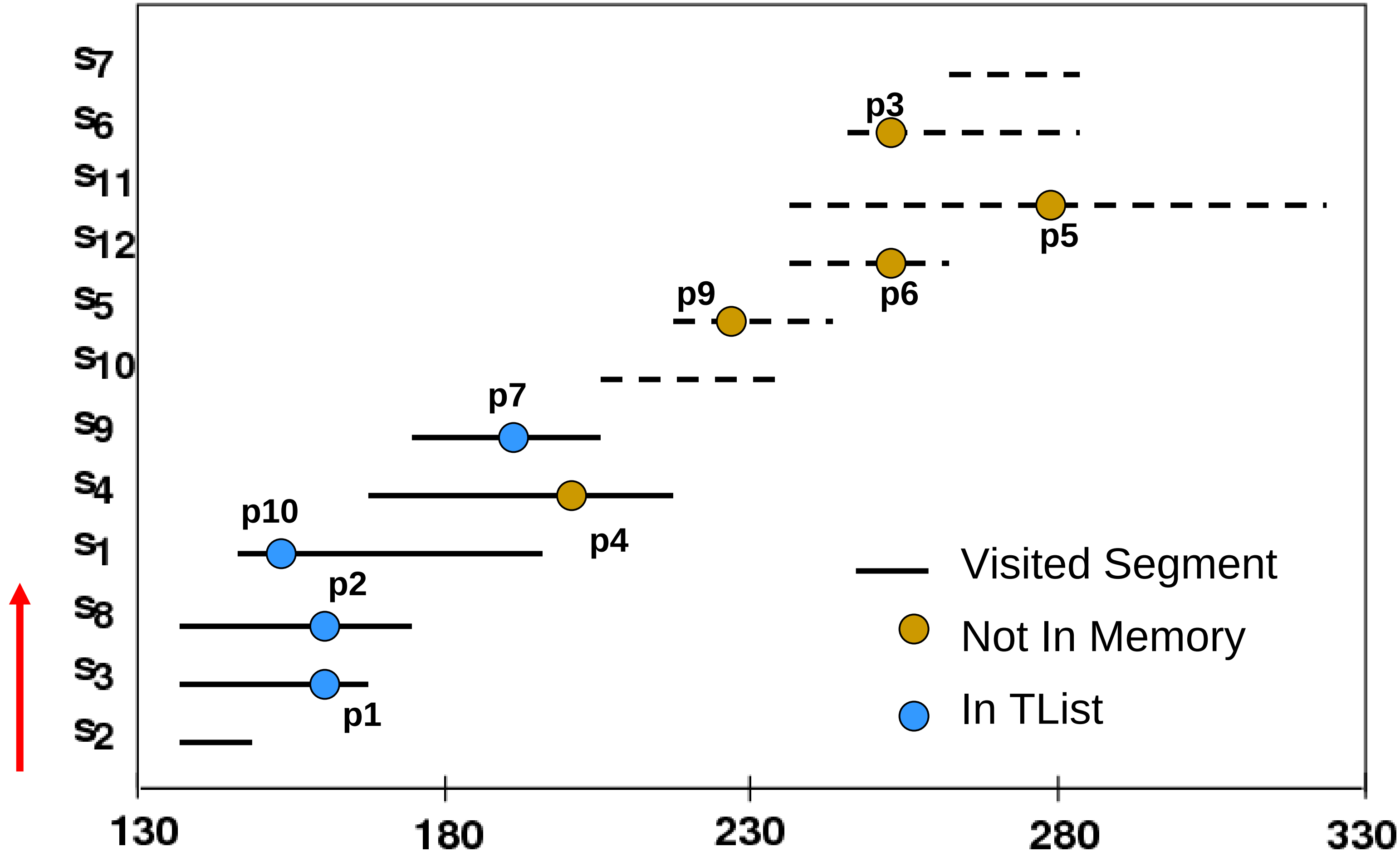
Naïve Monitoring Algorithm

Timestamp 2, remove p8, Kmax=Infinity, Visit s10?



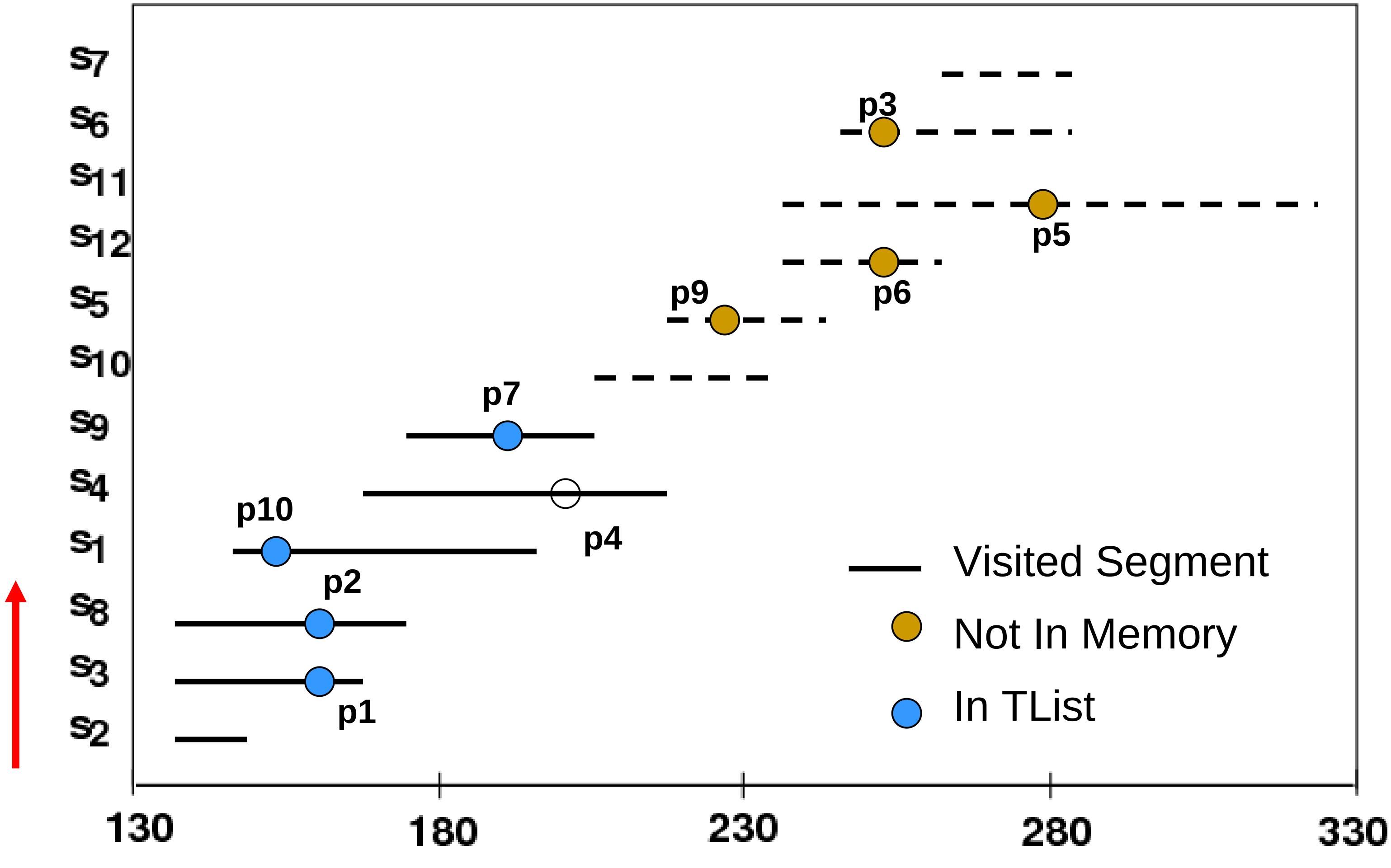
Naïve Monitoring Algorithm

Timestamp 2, remove p8: Re-computation



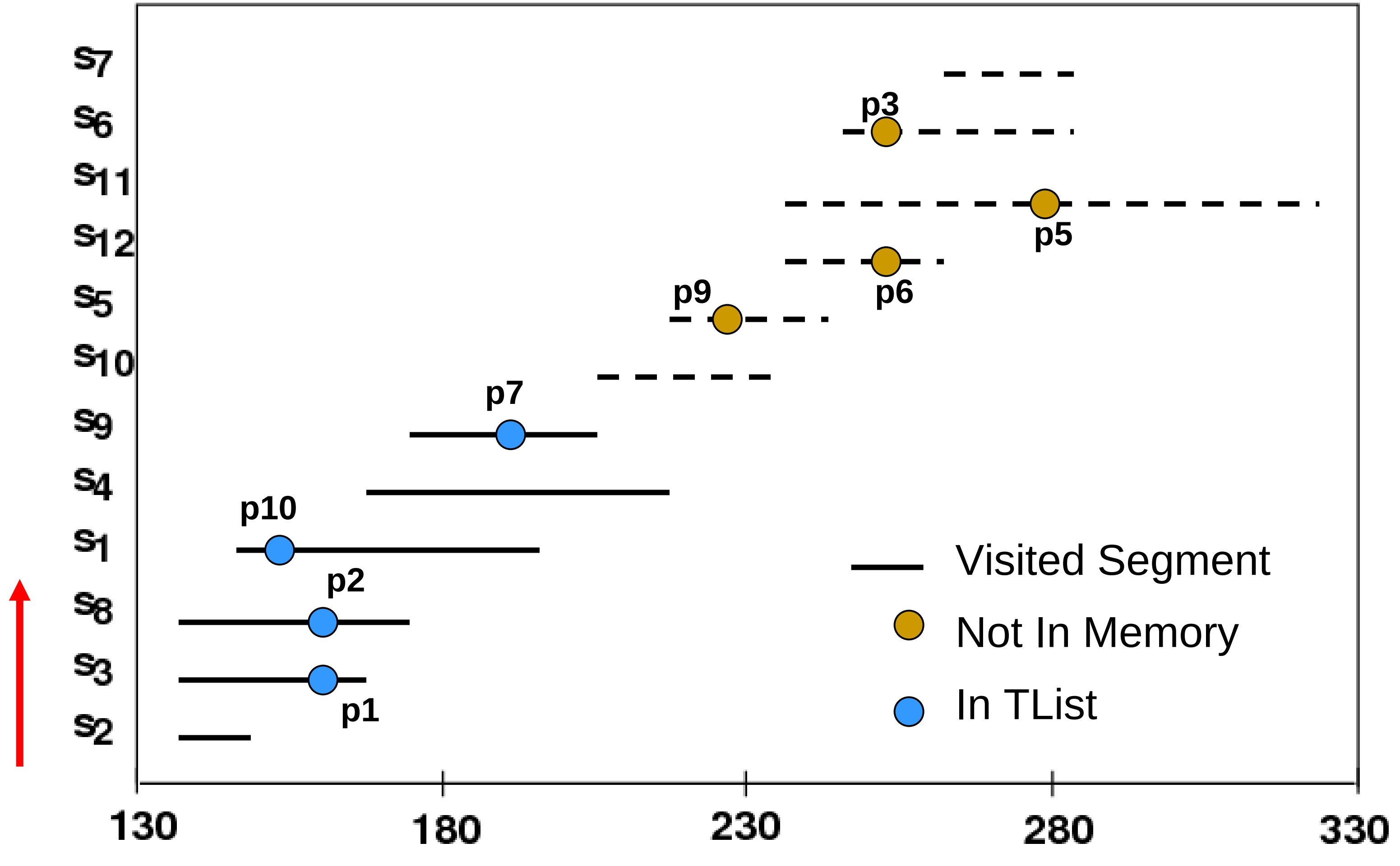
Naïve Monitoring Algorithm

Timestamp 3, remove p4



Naïve Monitoring Algorithm

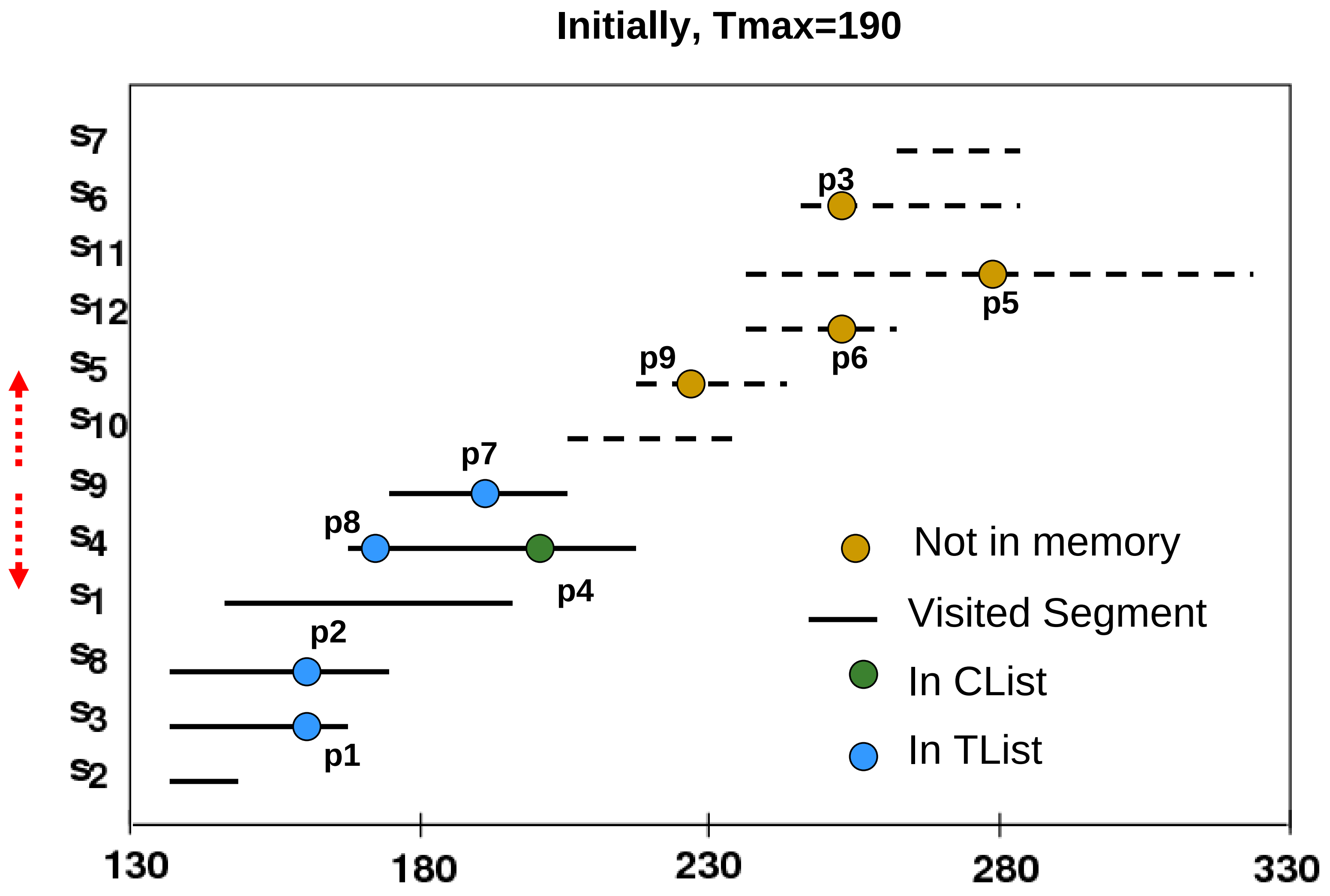
Timestamp 3, remove p4: Do nothing



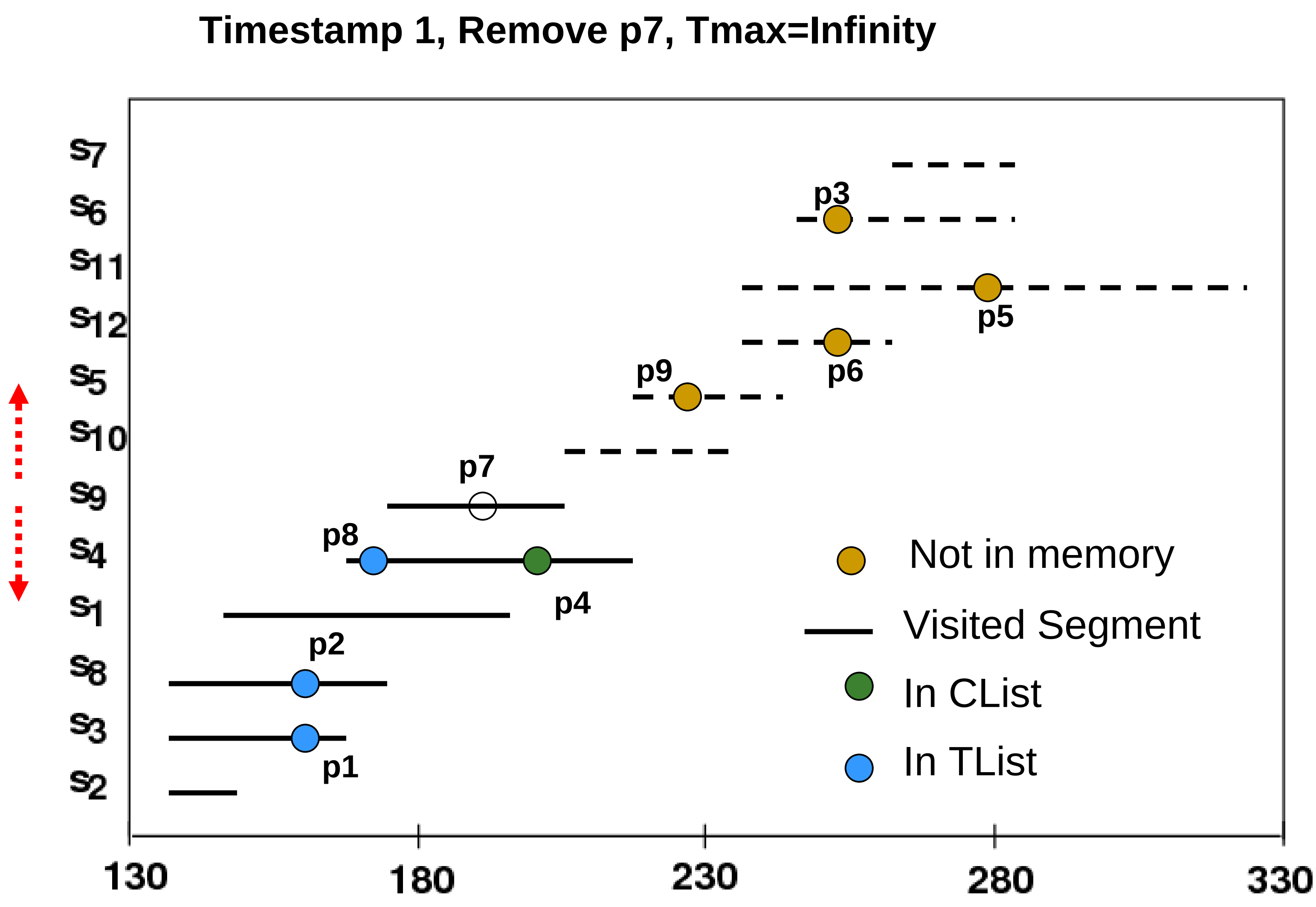
Bidirectional Updating Algorithm

- Sequential access: forward and backward
- CList: Candidate list
  - A List of Objects on the visited edges but not in TList
- Completely avoid re-computation
- Situations:
  - Add an object to TList: backward Update
  - Remove an object from TList: forward Update
  - Otherwise: update CList if necessary

Bidirectional Updating Algorithm

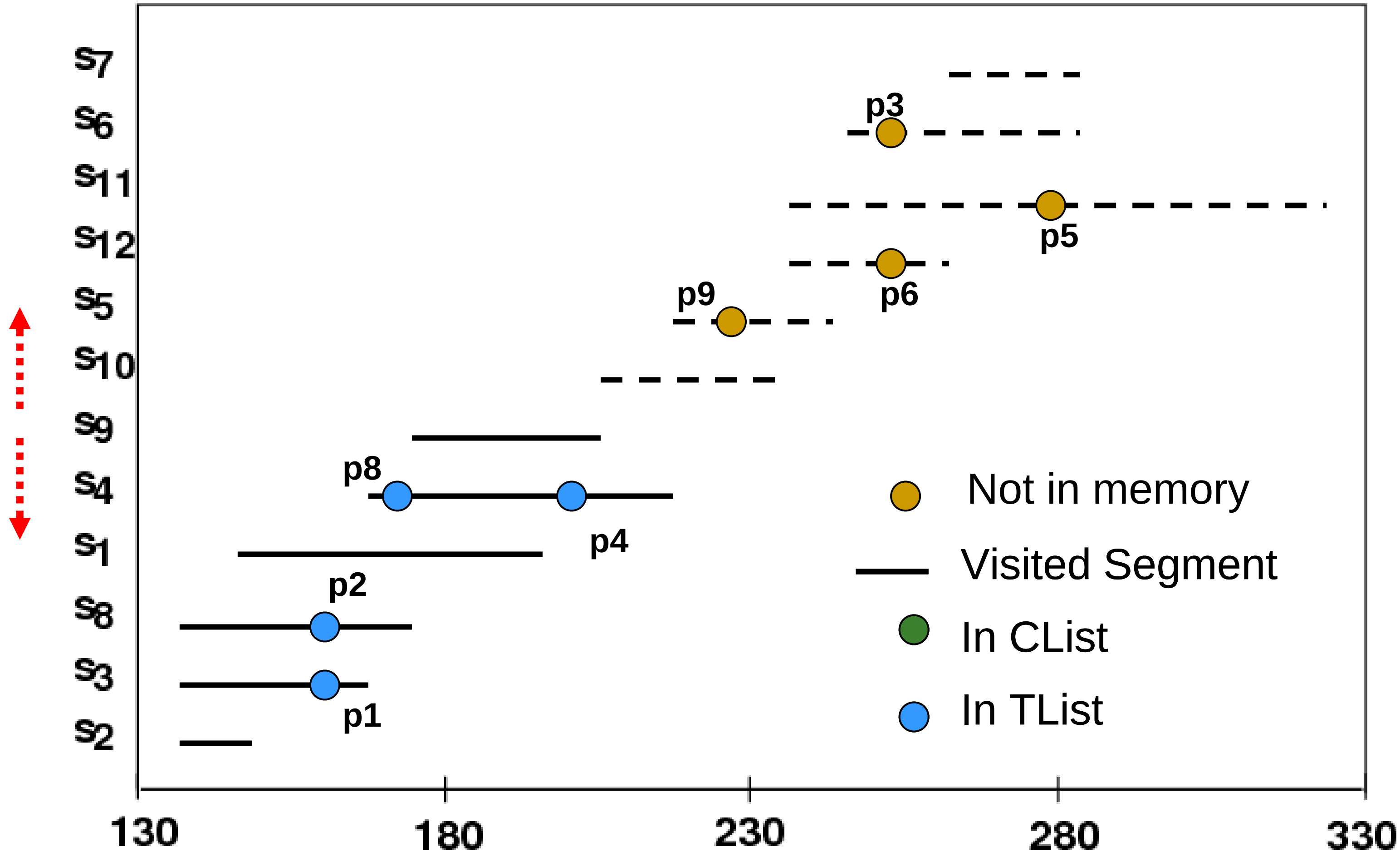


Bidirectional Updating Algorithm

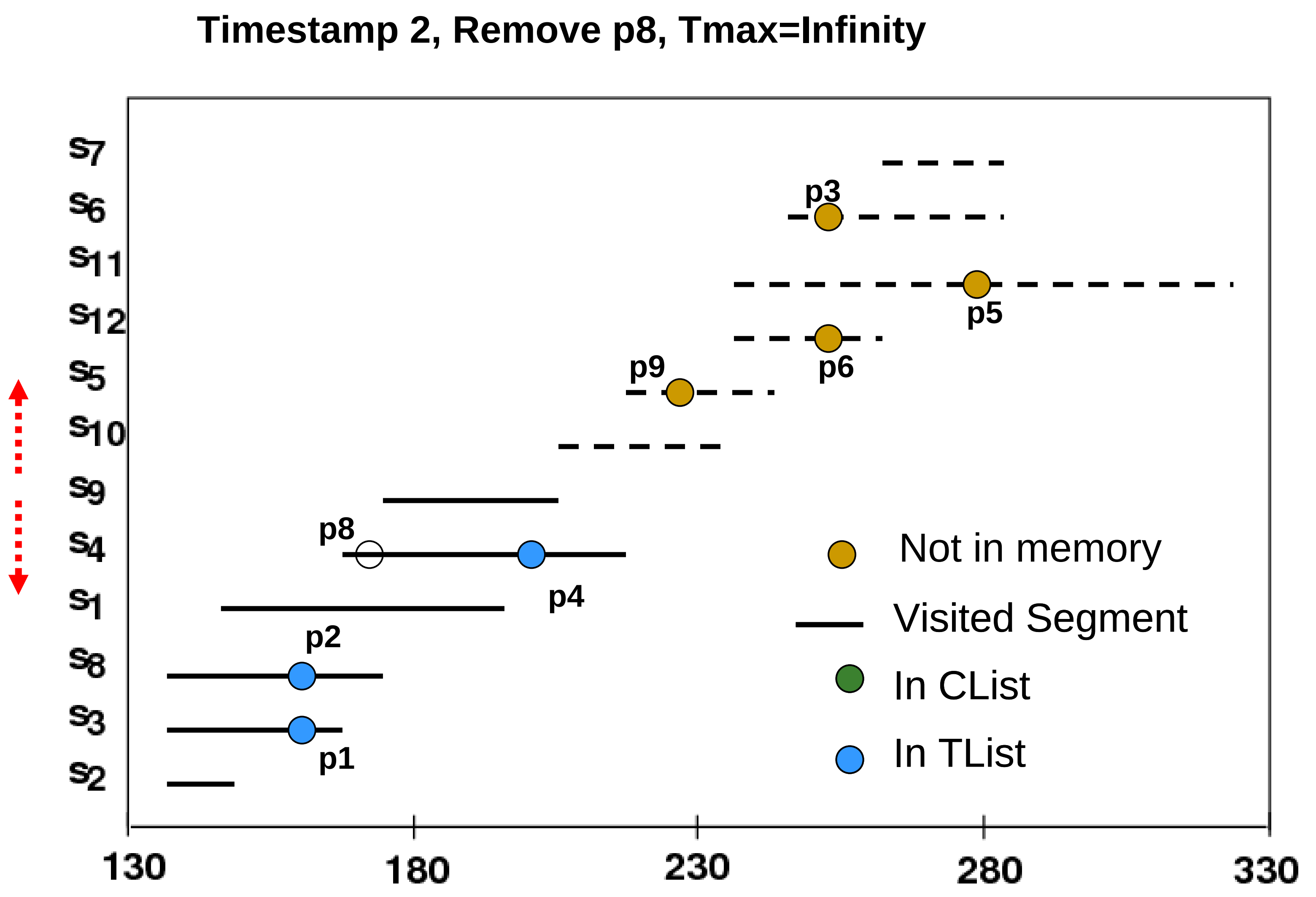


Bidirectional Updating Algorithm

Timestamp 1, Remove p7, Move p4 to TList, Tmax=200

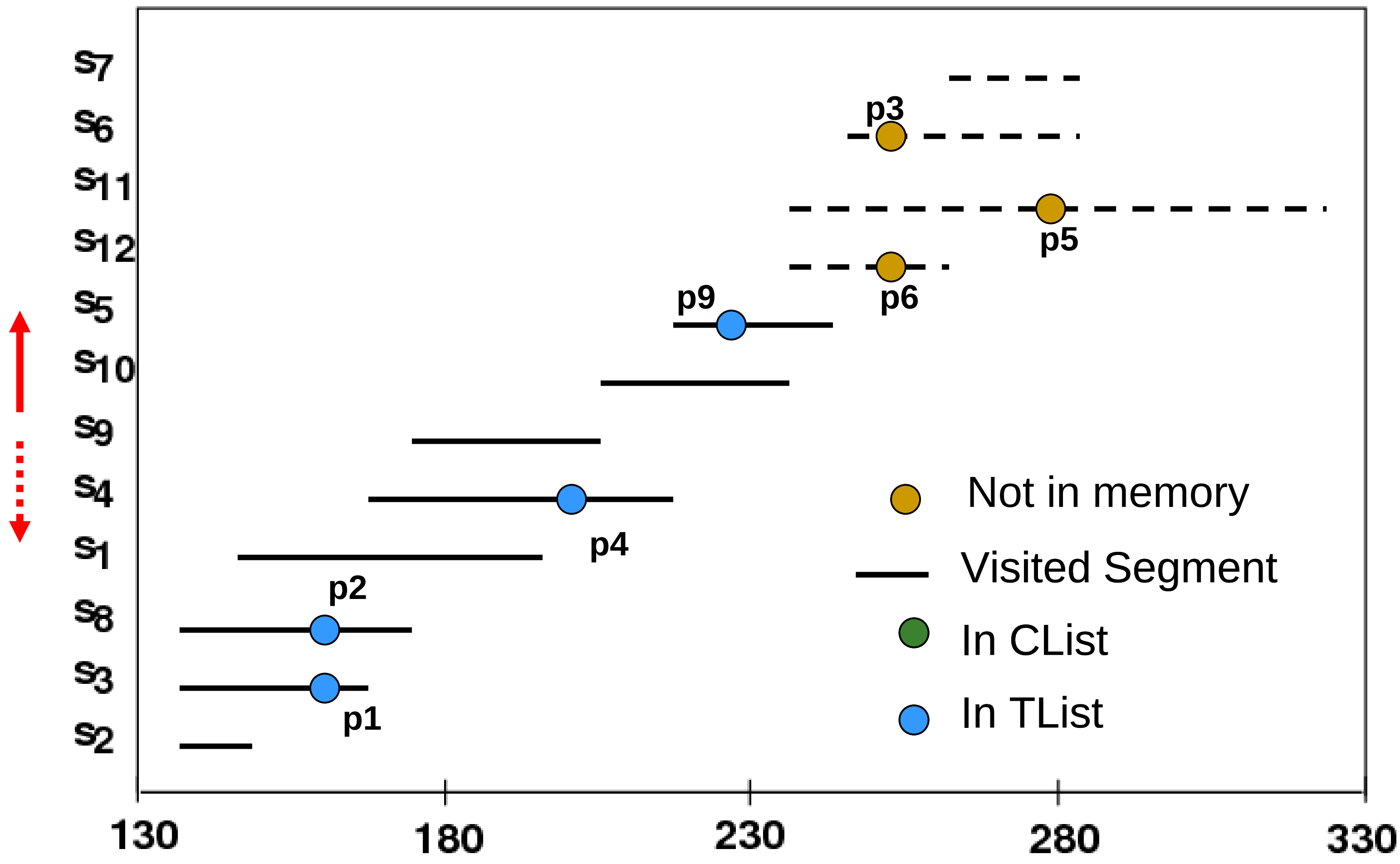


Bidirectional Updating Algorithm



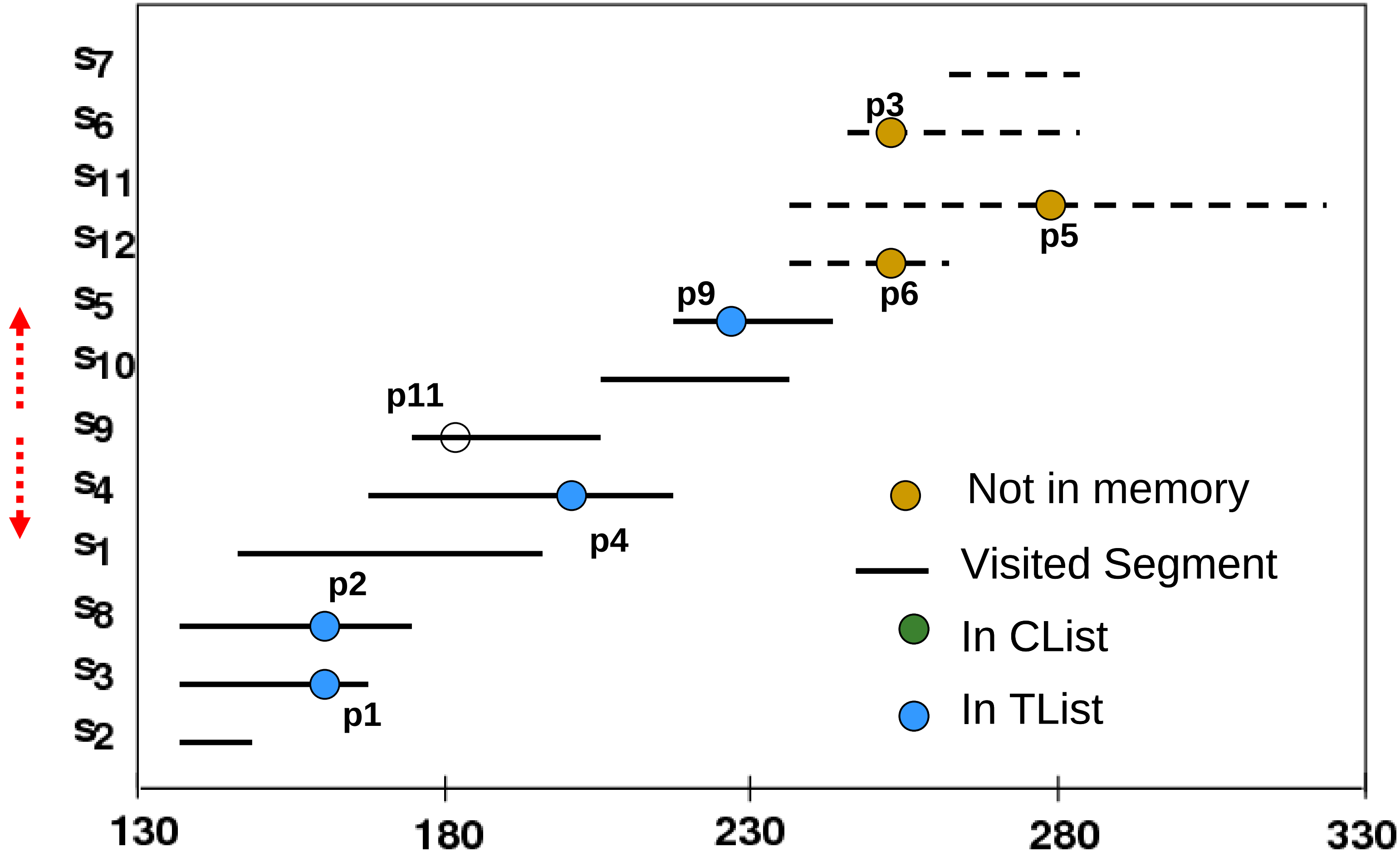
Bidirectional Updating Algorithm

Timestamp 2, Remove p8, Forward update, visit s10, s5, Tmax=230

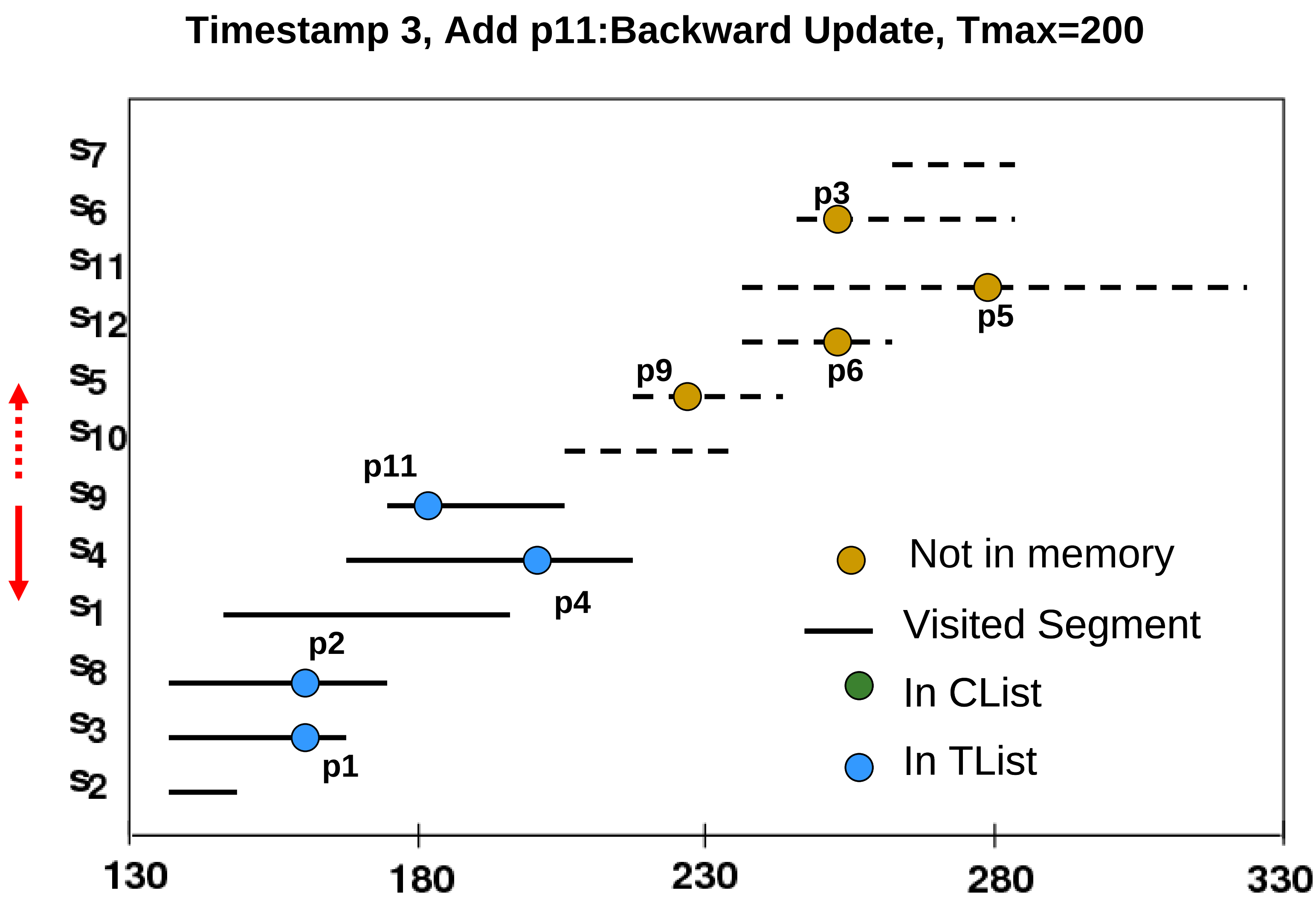


Bidirectional Updating Algorithm

Timestamp 3, Add p11



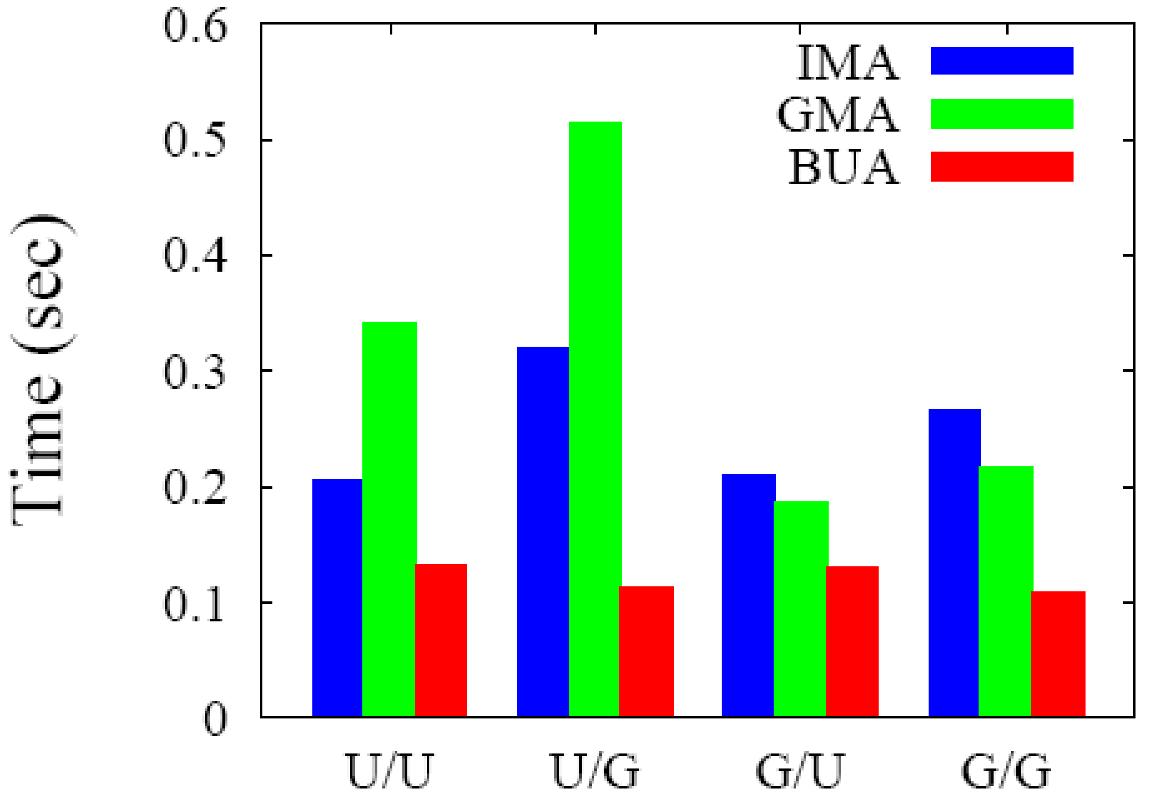
Bidirectional Updating Algorithm



Experiment Setup

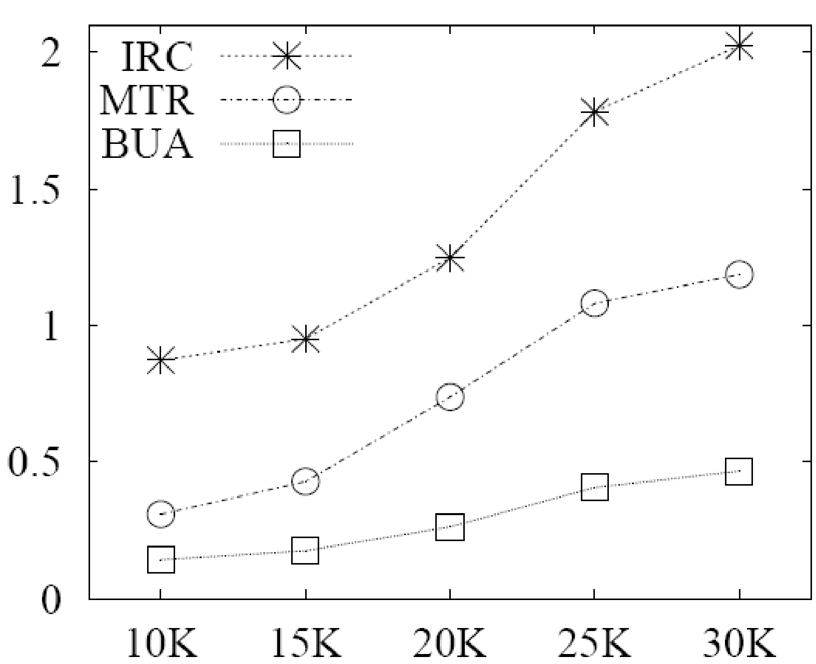
- We use road-map extracted from US Census Website.
- IRC: To compute the top-k results from scratch for every update.
- MTR: The Naïve monitoring algorithm
- BUA: Bidirectional updating algorithm
- One Query Point (VLDB06):
  - IMA: Incremental monitoring algorithm
  - GMA: Group monitoring algorithm

Experiment Result

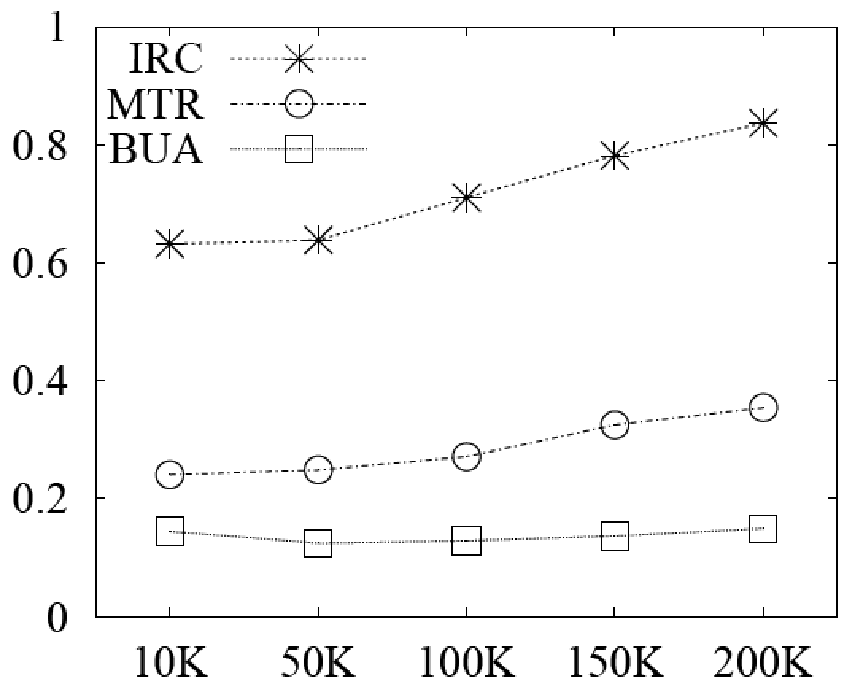


- Distribution of queries / Distribution of Objects
- U: Uniform G: Gaussian

Experiment Result: Test Network

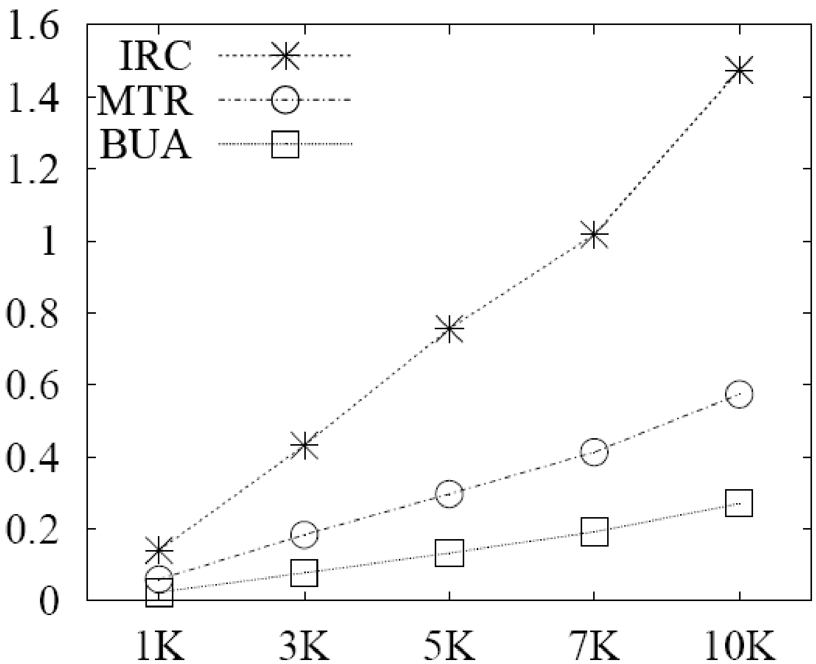


Vary Edges (Sec)

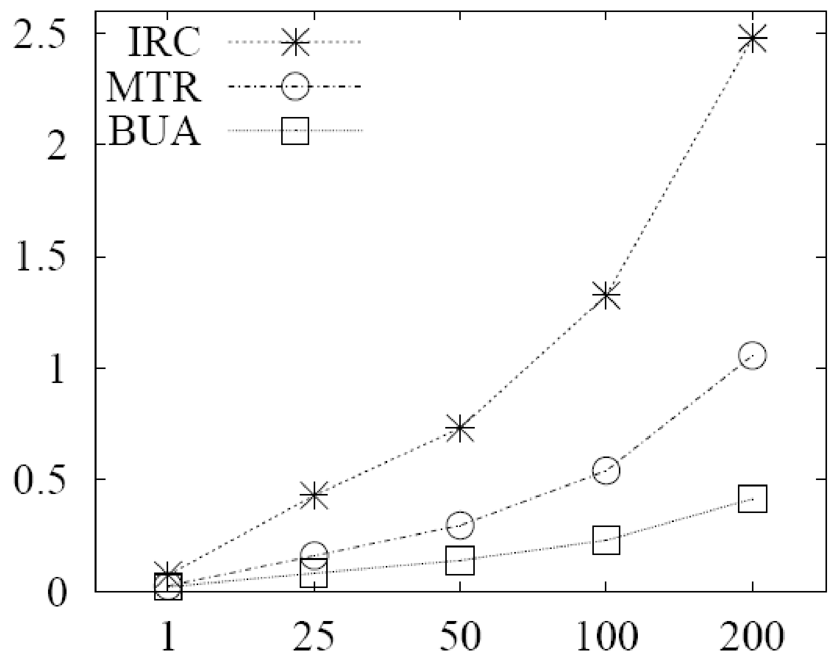


Vary Objects (Sec)

Experiment Result: Test Query



Vary Queries (Sec)



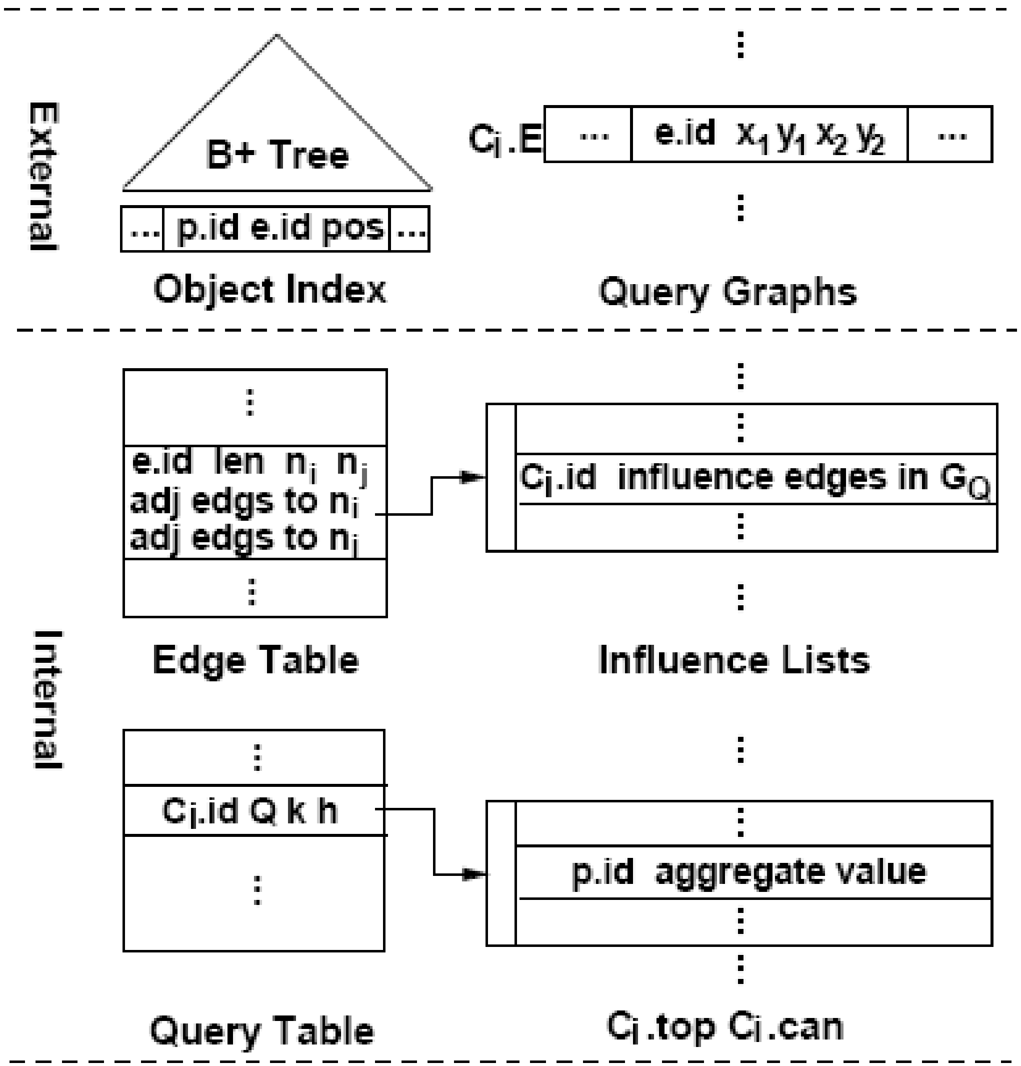
Vary k (Sec)

Summary

- Continuous Nearest Neighbor Query.
  - Monitor k-NN objects over a road network.
  - Minimize an aggregate distance function for multiple query points.
- Query Graph can be constructed offline.
- Bidirectional top-k monitoring algorithm to avoid re-computation.
- Extensive experiments are conducted using real road network maps.

Thank You!

Implementation Details



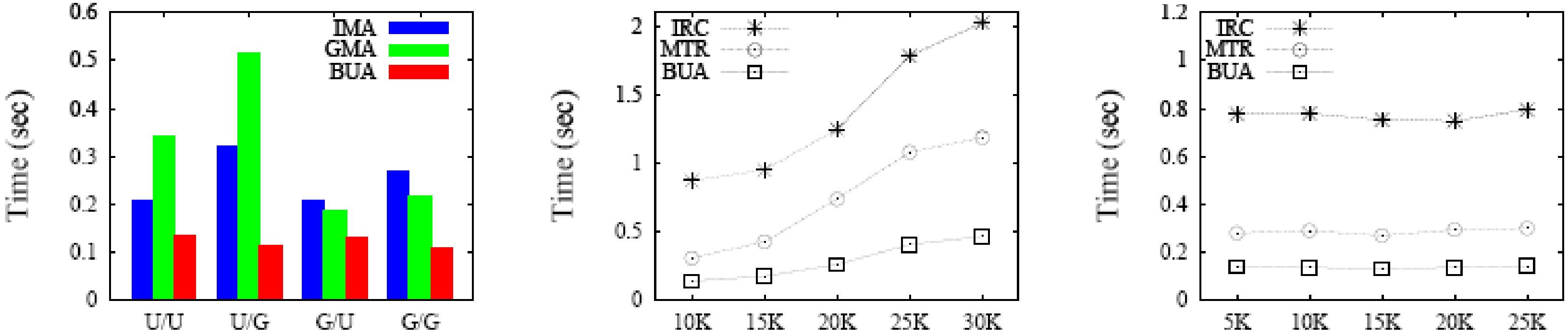
Experimental Studies

Parameter	Default	Range
Number of edges	25K	10, 15, 20, 25, 30 (K)
Number of nodes	20K	5, 10, 15, 20, 25 (K)
Number of queries	5K	1, 3, 5, 7, 10 (K)
Number of query points	20	1, 10, 20, 30, 40
Number of objects	100K	10, 50, 100, 150, 200 (K)
Query distribution	Uniform	Gaussian, Uniform
Object distribution	Uniform	Gaussian, Uniform
Top-k	50	1, 25, 50, 100, 200
Object agility	10%	5, 10, 15, 20, 25 (%)
Buffer size	2K	1, 2, 3, 4, 5 (K)
Function	SUM	MIN, MAX, SUM

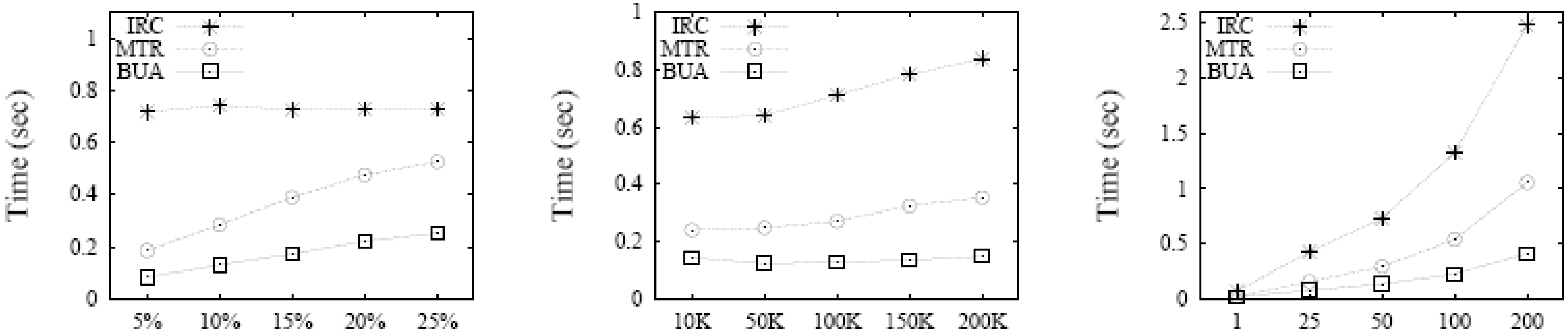
Experimental Studies

Time to construct query graph:

$\lvert E\rvert(K)$/T(ms)	10/114	15/214	20/319	25/408	30/505
$\lvert N\rvert(K)$/T(ms)	5/64	10/155	15/254	20/336	25/437
$\lvert Q\rvert$/T(ms)	1/26	10/178	20/343	30/506	40/675

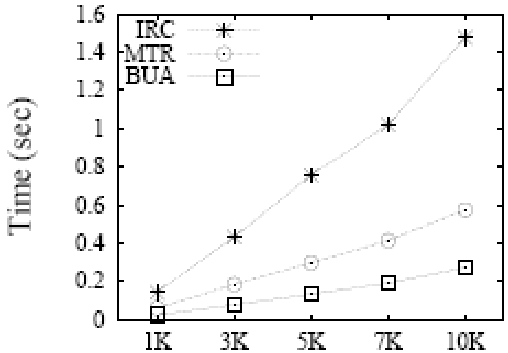


(a) IMA,GMA vs BUA (b) Vary Edges (c) Vary Nodes

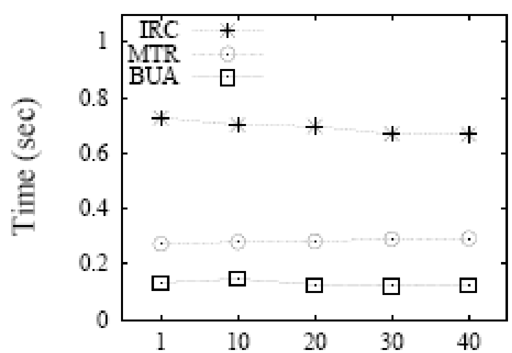


(d) Vary Object Agility (e) Vary Objects (f) Vary k

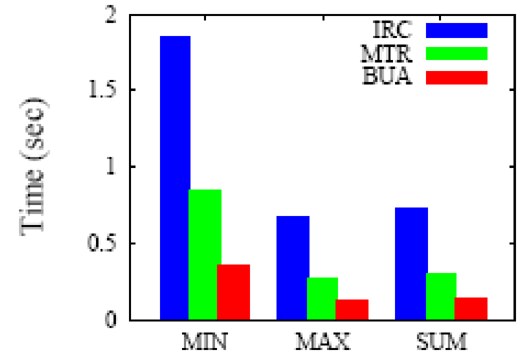
Experimental Studies



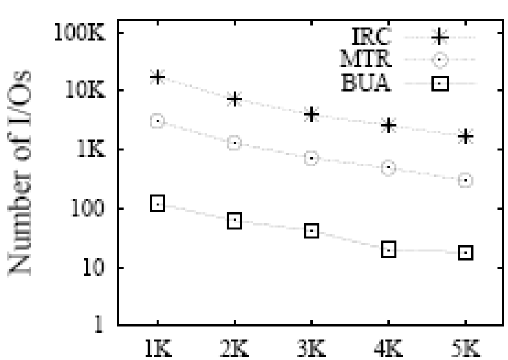
(g) Vary Queries



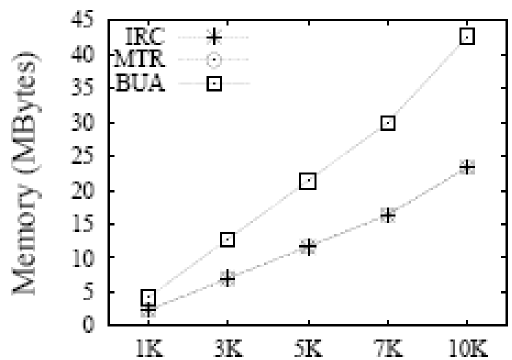
(h) Vary Query Points



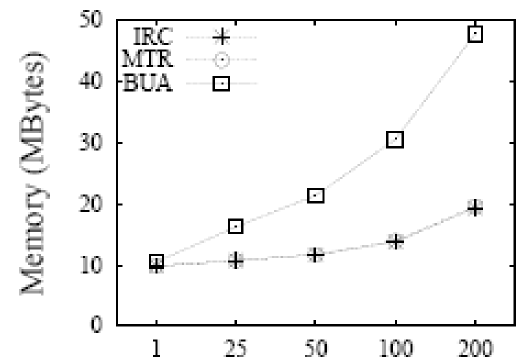
(i) Vary Functions



(j) Vary Buffer Size



(k) Vary Queries



(l) Vary k