

Approximation Strategies for Generalized Binary Search in Weighted Trees

Dariusz Dereniowski¹, Adrian Kosowski², Przemysław Uznański³,
Mengchuan Zou²

[1]Gdańsk University of Technology, Poland

[2]Inria Paris and IRIF, France

[3]ETH Zürich, Switzerland

ANR DESCARTES, Poitiers
Oct. 4th, 2017

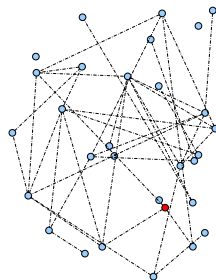
Content

- 1 Introduction
- 2 Preliminaries
- 3 Building a QPTAS
- 4 $O(\sqrt{\log n})$ -approximation algorithm
- 5 Conclusion and Perspective

Introduction

General Configuration of Searching Problem

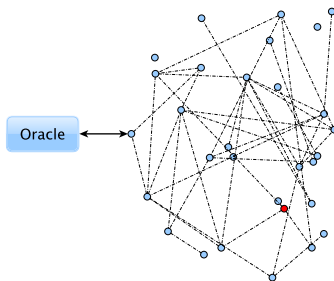
- A set of data organized in some structure
-
-



Introduction

General Configuration of Searching Problem

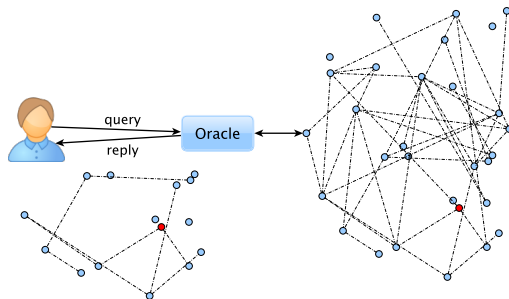
- A set of data organized in some structure
- An oracle replies to queries on the data
-



Introduction

General Configuration of Searching Problem

- A set of data organized in some structure
- An oracle replies to queries on the data
- The oracle returns a subset of the data set which contains the target element



Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

2	2	3	5	7	8	11	12	12	15	16	18
---	---	---	---	---	---	----	----	----	----	----	----



target x: 11

Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

2	2	3	5	7	8	11	12	12	15	16	18
---	---	---	---	---	---	----	----	----	----	----	----



target x: 11

Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

2	2	3	5	7	8	11	12	12	15	16	18
---	---	---	---	---	---	----	----	----	----	----	----



target x: 11

Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

2	2	3	5	7	8	11	12	12	15	16	18
---	---	---	---	---	---	----	----	----	----	----	----



target x: 11

Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

2	2	3	5	7	8	11	12	12	15	16	18
---	---	---	---	---	---	----	----	----	----	----	----



target x: 11

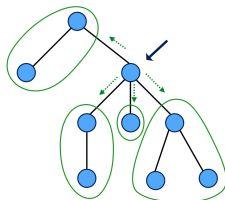
Generalized Binary Search in Trees

Binary Search

- For an ordered array (or totally ordered set)

Our problem : Searching in Trees

- Data organized into a tree
- Target node x is known to the oracle, but not to the search algorithm
- The oracle returns the subtree in which the target lies



Query Model for Trees

- **Query** : a node v



Query Model for Trees

- **Query** : a node v



- **Reply** :

- **true**, if v is the target



Query Model for Trees

– **Query** : a node v

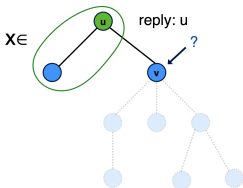


– **Reply** :

- **true**, if v is the target

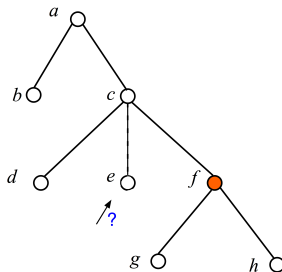


- otherwise, return **a neighbor u of v** which is closer to the target x



Example 1

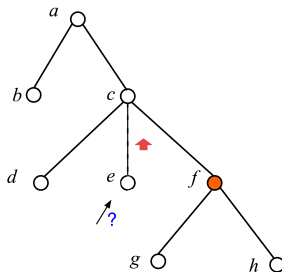
Query e



Target : f

Example 1

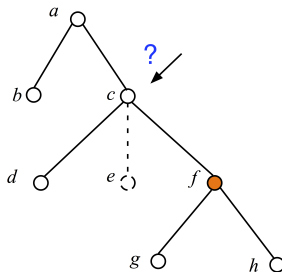
Query e



Target : f

Example 1

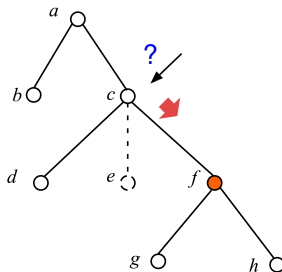
Query c



Target : f

Example 1

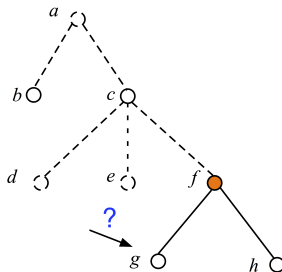
Query c



Target : f

Example 1

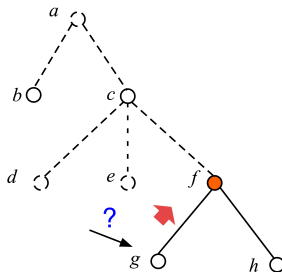
Query g



Target : f

Example 1

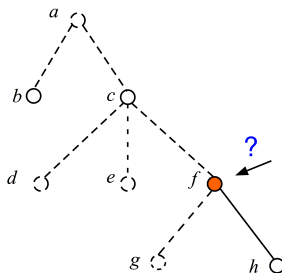
Query g



Target : f

Example 1

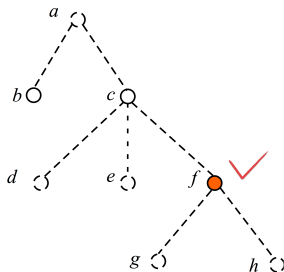
Query f



Target : f

Example 1

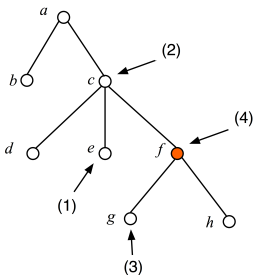
Found



Target : f

Example 1 : cost of locating the target

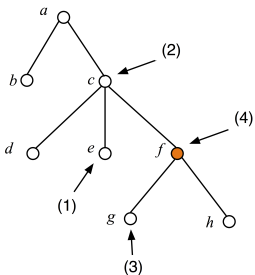
Unweighted variant



Total of 4 queries to locate target: e, c, g, f

Example 1 : cost of locating the target

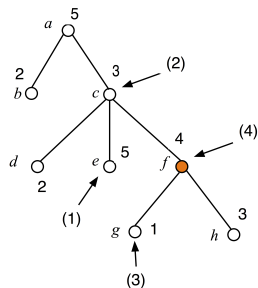
Unweighted variant



Total of 4 queries to locate target: e, c, g, f

Weighted variant

weight function $w: V \rightarrow \mathbb{R}_+$

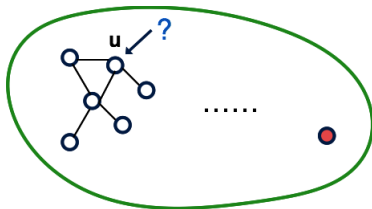


$$\begin{aligned} \text{Cost of locating target} &= w(e) + w(c) + w(g) + w(f) \\ &= 5 + 3 + 1 + 4 = 13 \end{aligned}$$

General Graph Variation

General Graph :

Query u

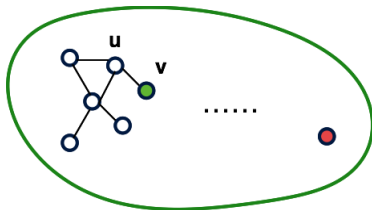


General Graph Variation

General Graph :

Query u

Reply a $v \in N(u)$, s.t. v is on the shortest path to the target



Search Strategy Problem in Trees

Setting

- Tree $T = (V, E, w)$ with root $r(T)$
- **Cost of query** to vertex v : $w: V \rightarrow \mathbb{R}_+$, $\max_v w(v) = 1$
- **Cost of search strategy** A on tree T : worst-case cost of finding a target
- **Optimal strategy** : search strategy with minimal cost on T , costs $OPT(T)$

Search Strategy Problem in Trees

Setting

- Tree $T = (V, E, w)$ with root $r(T)$
- **Cost of query** to vertex v : $w: V \rightarrow \mathbb{R}_+$, $\max_v w(v) = 1$
- **Cost of search strategy** A on tree T : worst-case cost of finding a target
- **Optimal strategy** : search strategy with minimal cost on T , costs $OPT(T)$

Our Problem :

- Input : tree $T = (V, E, w)$
- Compute : Optimal strategy for generalized binary search query model

Search Strategy Problem in Trees

Setting

- Tree $T = (V, E, w)$ with root $r(T)$
- **Cost of query** to vertex v : $w : V \rightarrow \mathbb{R}_+$, $\max_v w(v) = 1$
- **Cost of search strategy** A on tree T : worst-case cost of finding a target
- **Optimal strategy** : search strategy with minimal cost on T , costs $OPT(T)$

Our Problem :

- Input : tree $T = (V, E, w)$
- Compute : Optimal strategy for generalized binary search query model

Application Aspects

- Locating buggy nodes in network models
- Finding specific data in organized databases

State-of-the-Art

- Time complexity to compute optimal search strategy in different graphs

Graph class	unweighted	weighted
Path	$O(n)$ time	$O(n^2)$ time [1]
Tree	$O(n)$ time [2]	NP-hard [3]
Undirected Graph	$m^{\Theta(\log n)}$ under ETH [4]	PSPACE-complete [4]
Directed Graph	PSPACE-complete [4]	PSPACE-complete [4]

[1] Cicalese, Jacobs, Laber, Valentin, 2012

[2] Onak, Parys, 2006

[3] Dereniowski, Nadolski, 2006

[4] Emamjomeh-Zadeh, Kempe, Singhal, 2016

- Our scenario : Weighted trees

- NP-hard, $O(\log n)$ -approximation algorithm. [Dereniowski, 2006]
- $O\left(\frac{\log n}{\log \log \log n}\right)$ -approximation. [Cicalese, Jacobs, Laber, Valentin, 2012]
- $O\left(\frac{\log n}{\log \log n}\right)$ -approximation. [Cicalese, Keszezh, Lidický, Pálvölgyi, Valla, 2015]

Our Results

Results for the Search Strategy Problem in Weighted Trees :

Our Results

Results for the Search Strategy Problem in Weighted Trees :

Theorem 1. The problem admits a QPTAS.

- QPTAS : Quasi-Polynomial-Time Approximation Scheme, $(1 + \epsilon)$ -approximation algorithm running in $n^{\text{polylog}(n)}$ time, for any given $\epsilon > 0$.
- which implies that the problem is not APX-hard unless $NP \subseteq DTIME(n^{O(\log n)})$

Our Results

Results for the Search Strategy Problem in Weighted Trees :

Theorem 1. The problem admits a QPTAS.

- QPTAS : Quasi-Polynomial-Time Approximation Scheme, $(1 + \epsilon)$ -approximation algorithm running in $n^{\text{polylog}(n)}$ time, for any given $\epsilon > 0$.
- which implies that the problem is not APX-hard unless $NP \subseteq DTIME(n^{O(\log n)})$

Theorem 2. The problem admits a poly-time $O(\sqrt{\log n})$ -approximation algorithm.

- improves previous approximation ratio

[ICALP 2017, Dereniowski, Kosowski, Uznański, Zou]

Preliminaries : Characterization of a Valid Search Strategy

Characterization of a Search Strategy

- A query \Leftrightarrow an interval of time
 - length of interval : $l(v) = w(v)$
 - beginning time of v : when node v is queried during the search (if it is queried)
 - this time interval do not depend on the replies to other queries

Preliminaries : Characterization of a Valid Search Strategy

Characterization of a Search Strategy

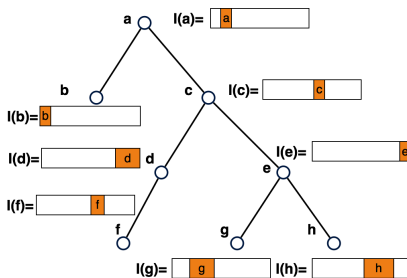
- A query \Leftrightarrow an interval of time
 - length of interval : $l(v) = w(v)$
 - beginning time of v : when node v is queried during the search (if it is queried)
 - this time interval do not depend on the replies to other queries

- A query sequence \Leftrightarrow intervals of time $l(v)$ for all $v \in V$

Preliminaries : Characterization of a Valid Search Strategy

Valid Search Strategy

- If the intervals of nodes u, v overlap, some node on the u - v path in the tree must be queried before both u and v .

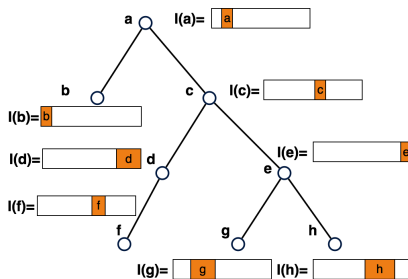


Extension of idea of : [Dereniowski, 2006]

Preliminaries : Characterization of a Valid Search Strategy

Valid Search Strategy

- If the intervals of nodes u, v overlap, some node on the u - v path in the tree must be queried before both u and v .



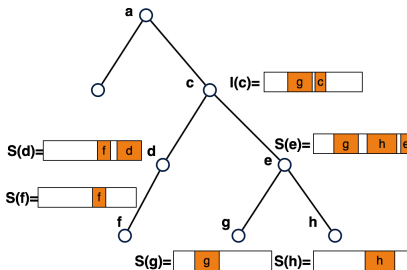
Lemma. There is a equivalence between optimizing search strategy and the following problem :

Assign intervals $I(v) = [a, b]$ to $v \in V$, where $|[a, b]| = w(v)$, s.t. $\forall u, v \in V$, $I(u) \cap I(v) \neq \emptyset \Rightarrow \exists z$ on the path from u to v and $\max(I(z)) \leq \min(I(u), I(v))$

Schedule assignments

Schedule assignments :

- Schedule $S(v)$: set of time intervals of "uncovered" nodes in the subtree of v
- The interval $I(u)$ is "covered" by an ancestor x if x is assigned an earlier interval i.e. $(\max I(x) \leq \min I(u))$.
- Schedule assignment : schedule $S(v)$ for all $v \in V$



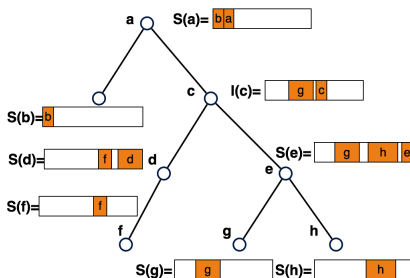
Constraints

- No two interval in the schedule of a node could overlap

Schedule assignments

Schedule assignments :

- Schedule $S(v)$: set of time intervals of "uncovered" nodes in the subtree of v
- The interval $I(u)$ is "covered" by an ancestor x if x has a strictly earlier interval, e.g. ($\max I(x) < \min I(u)$).
- Schedule assignment : schedule $S(v)$ for all $v \in V$



Rounding and Alignment

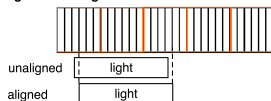
Units of time : box and slot

- A box : time between integer multiples of $\frac{\varepsilon}{\log n}$
- A slot : time between integer multiples of $\frac{\varepsilon}{n}$



Rounding and Alignment : Time intervals of vertices are rounded and aligned to boxes or slots depending on their weight ("heavy" : $w(v) > \frac{1}{\log n}$, "light" : $w(v) \leq \frac{1}{\log n}$)

light vertex aligned to slots



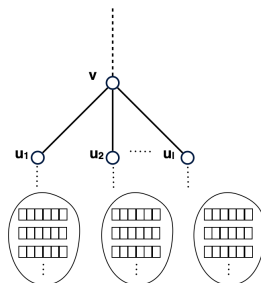
heavy vertex aligned to boxes



Lemma. Given tree T , there is an aligned schedule assignment S' with $cost_{S'}(T) \leq (1 + 11\varepsilon)OPT(T)$

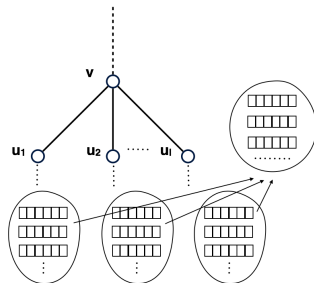
Dynamic Programming

- Assume node v has children u_1, \dots, u_l
- Store all valid aligned schedules $S(v)$ for every node
- Compute in bottom-up manner :



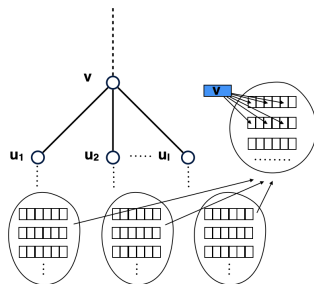
Dynamic Programming

- Assume node v has children u_1, \dots, u_l
- Store all valid aligned schedules $S(v)$ for every node
- Compute in bottom-up manner :
 - 1 Construct all valid aligned schedules for children u_1, \dots, u_l



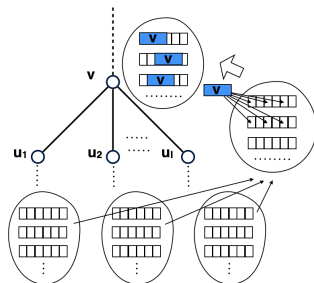
Dynamic Programming

- Assume node v has children u_1, \dots, u_l
- Store all valid aligned schedules $S(v)$ for every node
- Compute in bottom-up manner :
 - 1 Construct all valid aligned schedules for children u_1, \dots, u_l
 - 2 Enumerate all possible start times of v



Dynamic Programming

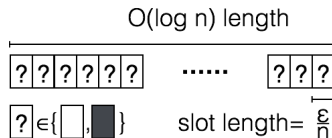
- Assume node v has children u_1, \dots, u_l
- Store all valid aligned schedules $S(v)$ for every node
- Compute in bottom-up manner :
 - 1 Construct all valid aligned schedules for children u_1, \dots, u_l
 - 2 Enumerate all possible start times of v
 - 3 Obtain all possible schedules for v



Running Time

Fact. We have $1 \leq OPT(T) \leq \lceil \log_2 n \rceil$.

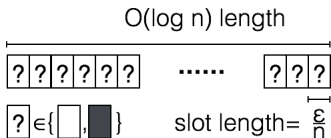
\Rightarrow In Dynamic Programming, only consider aligned schedules of duration $\leq O(\log n)$.



Running Time

Fact. We have $1 \leq OPT(T) \leq \lceil \log_2 n \rceil$.

\Rightarrow In Dynamic Programming, only consider aligned schedules of duration $\leq O(\log n)$.



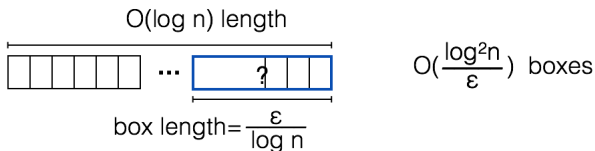
$O\left(\frac{n}{\epsilon} \log n\right)$ slots

$2^{O\left(\frac{n}{\epsilon} \log n\right)}$ possibilities

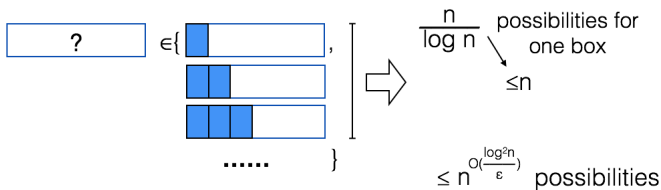
Exponential!

Speed up Running Time

Relaxation : disregarding order of queries strictly inside a box



For each box, store only number of full slots in this box.



Speed up Running Time

Relaxation computed by dynamic programming :

- can be computed exactly in $n^{O(\frac{\log^2 n}{\epsilon})}$ time
 - * running time can be reduced to $n^{O(\frac{\log n}{\epsilon^2})}$ by adaptively choosing box sizes
- not worse cost than optimal schedule
- not a valid schedule assignment
 - * but : can be fixed at small extra cost [non-trivial, based on solution of optimal strategy in unweighted trees]


QPTAS

- \hat{S}^* – solution by DP routine disregarding orders of light queries strictly inside a box.
- R – an subsequence of nodes based on optimal solution for unweighted trees
- S^+ – a valid $(1 + \varepsilon)$ approximation



QPTAS

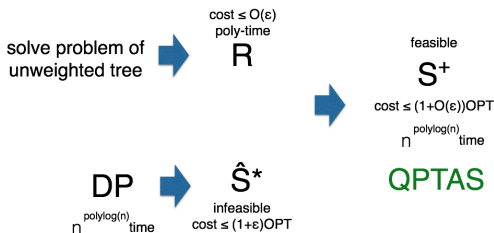
- \hat{S}^* – solution by DP routine disregarding orders of light queries strictly inside a box.
- R – an subsequence of nodes based on optimal solution for unweighted trees
- S^+ – a valid $(1 + \varepsilon)$ approximation

solve problem of unweighted tree  R
 cost $\leq O(\varepsilon)$
 poly-time

DP  \hat{S}^*
 $n^{\text{poly}(\log(n))}$ time infeasible
 cost $\leq (1+\varepsilon)\text{OPT}$

QPTAS

\hat{S}^* – solution by DP routine disregarding orders of light queries strictly inside a box.
 R – an subsequence of nodes based on optimal solution for unweighted trees
 S^+ – a valid $(1 + \varepsilon)$ approximation



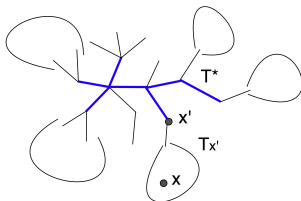
From QPTAS to $O(\sqrt{\log n})$ -approximation algorithm

Corollary of QPTAS

- set $\varepsilon = 1 \Rightarrow n^{O(\log n)}$ running time constant-factor approximation

Recursive decomposition with central subtree T^*

- Cost on $T = \text{Cost of locating } x' \text{ in } T^* + \text{Cost of executing the strategy in } T_{x'}$



From QPTAS to $O(\sqrt{\log n})$ -approximation algorithm

Corollary of QPTAS

- set $\varepsilon = 1 \Rightarrow n^{O(\log n)}$ running time constant-factor approximation

Recursive decomposition with central subtree T^*

- Cost on $T = \text{Cost of locating } x' \text{ in } T^* + \text{Cost of executing the strategy in } T_{x'}$

Result

- We can choose T^* , such that :
 - Running time of every recursion level is $\text{poly}(n)$, constant factor approximation
 - Recursion depth is bounded by $O(\sqrt{\log n})$
- Approximation factors add up along recursions
- Results in a polynomial-time $O(\sqrt{\log n})$ -approximation algorithm

Conclusion

Main Results

- A QPTAS (quasi-polynomial-time approximation scheme) for strategies of generalized binary search in weighted trees
 - implies the problem is not APX-hard unless $NP \subseteq DTIME(n^{O(\log n)})$
- An $O(\sqrt{\log n})$ -approximation polynomial-time algorithm for strategies of generalized binary search in weighted trees
 - improves previous approximation ratio

Open Questions

- Find constant-factor approximation ?
- Results for other classes of graphs ?
- Oracle with error-reply rate ?

Conclusion

Main Results

- A QPTAS (quasi-polynomial-time approximation scheme) for strategies of generalized binary search in weighted trees
 - implies the problem is not APX-hard unless $NP \subseteq DTIME(n^{O(\log n)})$
- An $O(\sqrt{\log n})$ -approximation polynomial-time algorithm for strategies of generalized binary search in weighted trees
 - improves previous approximation ratio

Open Questions

- Find constant-factor approximation ?
- Results for other classes of graphs ?
- Oracle with error-reply rate ?

Thanks !

Thanks

Thanks!