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, Poitier
Oct. 4th, 2017
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)

Example:

```
2 2 3 5 7 8 11 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 $v$

[Diagram of a node $v$ with an arrow pointing to it]
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**

- 4 queries to locate the target: e, c, g, f

**Weighted variant**

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

- Cost of locating the target = $w(e)+w(c)+w(g)+w(f)$
  $= 5+3+1+4=13$
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
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

<table>
<thead>
<tr>
<th>Graph class</th>
<th>unweighted</th>
<th>weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>O(n) time</td>
<td>O(n^2) time [1]</td>
</tr>
<tr>
<td>Tree</td>
<td>O(n) time</td>
<td>NP-hard [3]</td>
</tr>
<tr>
<td>Undirected Graph</td>
<td>m^Θ(log n) under ETH [4]</td>
<td>PSPACE-complete [4]</td>
</tr>
</tbody>
</table>


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

Results for the Search Strategy Problem in Weighted Trees:

Theorem 1. The problem admits a QPTAS.

QPTAS: Quasi-Polynomial-Time Approximation Scheme, \( O(\sqrt{\log n}) \)-approximation algorithm running in \( n \text{polylog}(n) \) time, for any given \( \varepsilon > 0 \).

This implies that the problem is not APX-hard unless \( \text{NP} \subseteq \text{DTIME}(n^{O(\log n)}) \).

Theorem 2. The problem admits a poly-time \( O(p \log n) \)-approximation algorithm.

Improves previous approximation ratio [ICALP 2017, Dereniowski, Kosowski, Uznański, Zou].
Our Results

Results for the Search Strategy Problem in Weighted Trees:

Theorem 1. The problem admits a QPTAS.

- QPTAS: Quasi-Polynomial-Time Approximation Scheme, $(1 + \varepsilon)$-approximation algorithm running in $n^{\text{polylog}(n)}$ time, for any given $\varepsilon > 0$.

- which implies that the problem is not APX-hard unless $NP \subseteq DTIME(n^{O(\log n)})$. 

\[O(\sqrt{\log n})\]-approximation algorithm
Our Results

Results for the Search Strategy Problem in Weighted Trees:

Theorem 1. The problem admits a QPTAS.

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

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 $v \in V$
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]
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 an equivalence between optimizing search strategy and the following problem:

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

- Schedule $S(v)$: set of time intervals of "uncovered" nodes in the subtree of $v$
- The interval $l(u)$ is "covered" by an ancestor $x$ if $x$ is assigned an earlier interval i.e. $(\max l(x) \leq \min l(u))$.
- 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 \( S(\nu) \): set of time intervals of "uncovered" nodes in the subtree of \( \nu \)
- The interval \( l(u) \) is "covered" by an ancestor \( x \) if \( x \) a strictly earlier interval, e.g. \((\max l(x) \leq \min l(u))\).
- Schedule assignment: schedule \( S(\nu) \) for all \( \nu \in V \)
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": $w(v) \leq \frac{1}{\log n}$)

**Lemma.** Given tree $T$, there is an aligned schedule assignment $S'$ with $\text{cost}_{S'}(T) \leq (1 + 11\varepsilon)\text{OPT}(T)$
Dynamic Programming

- Assume node $v$ has children $u_1, \ldots, u_l$

- Store all valid aligned schedules $S(v)$ for every node

- Compute in bottom-up manner:
Dynamic Programming

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

- Assume node \( v \) has children \( u_1, \ldots, u_l \)

- Store all valid aligned schedules \( S(v) \) for every node

- Compute in bottom-up manner:
  1. Construct all valid aligned schedules for children \( u_1, \ldots, u_l \)
  2. Enumerate all possible start times of \( v \)
Dynamic Programming

- Assume node $v$ has children $u_1, \ldots, u_l$

- Store all valid aligned schedules $S(v)$ for every node

- Compute in bottom-up manner:
  1. Construct all valid aligned schedules for children $u_1, \ldots, u_l$
  2. Enumerate all possible start times of $v$
  3. Obtain all possible schedules for $v$
Running Time

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

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

$O(\log n)$ length

\[ \square \square \square \square \square \quad \cdots \quad \square \square \square \]

\[ ? \in \{ [\square, \square] \} \quad \text{slot length} = \frac{\varepsilon}{n} \]
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(log n) length**
  - $? \in \{?, \square\}$
  - slot length = $\frac{\epsilon}{n}$

- **O($\frac{n}{\epsilon}$ log n) slots**

- **$2^{O(\frac{n}{\epsilon} \log n)}$ possibilities**

**Exponential!**
Speed up Running Time

Relaxation: disregarding order of queries strictly inside a box

\[ O(\log n) \text{ length} \]

\[ \text{box length} = \frac{\varepsilon}{\log n} \]

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

\[ \frac{n}{\log n} \text{ possibilities for one box} \leq n \]

\[ \leq n^{O\left(\frac{\log^2 n}{\varepsilon}\right)} \text{ possibilities} \]
Speed up Running Time

Relaxation computed by dynamic programming:

- can be computed exactly in $n^{O\left(\frac{\log^2 n}{\varepsilon}\right)}$ time
  * running time can be reduced to $n^{O\left(\frac{\log n}{\varepsilon^2}\right)}$ 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]
$\hat{S}^*$ – 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

\[\text{DP} \quad \text{infeasible cost} \leq (1+\varepsilon)\text{OPT}\]
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
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