Optimal placement of storage nodes in a wireless sensor network

Gianlorenzo D’Angelo\textsuperscript{1}
Daniele Diodati\textsuperscript{2}    Alfredo Navarra\textsuperscript{2}    Cristina M. Pinotti\textsuperscript{2}

1 - Gran Sasso Science Institute
2 - University of Perugia
Scenario

- Given a wireless sensor network represented as a graph
- And a special *sink* node $r$
- All the sensors collect data with a regular frequency and send them to $r$ along the shortest paths
• Alternatively the data can be forwarded to some *storage* nodes
• Storage nodes *compress* and *aggregate* the data, and then send them to the sink (reduced in size)
Given a fixed integer $k$, how to choose the “best” $k$ storage nodes among the nodes of the network in order to minimize the energy consumption?
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
Given:

- a weighted connected graph \( G = (V, E, w) \) representing a wireless sensor network where each \( v \in V \) generates raw data with size \( s_d(v) \)
- an integer \( k \).

We aim at finding a set \( S \subseteq V \) of storage nodes such that \( |S| \leq k \)

- Each \( v \in V \) is associated to a storage node, denoted as \( \sigma(v) \in S \)
- In \( \sigma(v) \), the compressed size of the data produced by a node \( v \) becomes \( \alpha s_d(v) \), with \( \alpha \in [0, 1] \)

Total cost: \( \text{cost}(S) = \sum_{v \in V} s_d(v) (d(v, \sigma(v)) + \alpha d(\sigma(v), r)) \)
For $v = 2$ the cost is:

$$s_d(2) \cdot (w(2, 7) + w(7, 3)) + \alpha \cdot s_d(2) \cdot (w(3, 6), w(6, r))$$

Total cost: $cost(S) = \sum_{v \in V} s_d(v) (d(v, \sigma(v)) + \alpha d(\sigma(v), r))$

The minimum $k$-storage problem (briefly, MSP) consists in finding a subset $S \subseteq V$, with $|S| \leq k$ that minimizes $cost(S)$
Related Work

- [Sheng et al. 2007] 10-approximation algorithm for the case
  - $s_d(v)$ is a constant for any $v$
  - The distances are given by Euclidean distances
- [Sheng et al. 2010] Optimal algorithms for trees
  - Either limited or unlimited $k$
  - They consider the cost of diffusing the query
  - The algorithms are polynomial only if the degree of the tree is bounded
Our results

- Polynomial-time exact algorithms
  - For trees in directed graphs
  - For bounded-treewidth undirected graphs
- Approximation lower bounds
  - Not in $APX$ in directed graphs
  - $1 + \frac{1}{e} > 1.367$ for undirected graph
- Local search algorithm for undirected graphs with constant approximation ratio
- Experimental evaluation of such algorithm on several graph topologies
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
Directed trees

Idea: Transform the generic rooted tree into an equivalent binary tree
We devise a dynamic programming algorithm for binary trees

**Theorem**

Given a directed tree $T$, there exists an algorithm that optimally solves MSP in $O(\min\{kn^2, k^2P\})$, where $P$ is the path-length of $T$.

Path-length: Sum over the whole tree of the number of arcs on the path from each tree node to the root

- Balanced binary tree: $P = \Theta(n \log n)$,
- Random tree: $P = \Theta(n\sqrt{n})$
- Worst case: $P = O(n^2)$
We exploit the concept of tree decomposition to devise a dynamic programming algorithm.

**Theorem**

Given an undirected graph $G$ and a tree-decomposition of $G$ with width $w$, there exists an algorithm that optimally solves MSP in $O(\cdot k \cdot n^{w+3})$ time.
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
We show that $MSP$ in undirected graphs cannot be approximated within a factor of $1 + \frac{1}{e}$, unless $P = NP$

In detail,

- We show that the metric $k$-median problem cannot be approximated within a factor of $1 + \frac{1}{e}$, unless $P = NP$
- We show that $MSP$ is at least as hard to approximate as the metric $k$-median problem
The metric $k$-median problem

Let

- $G = (V, E)$ be a complete graph
- $k \in \mathbb{N}$
- $\text{dist}(u, v) \in \mathbb{N}$ be the distance from $u$ to $v$ over the edge $(u, v) \in E$

A $k$-median set for $G$ is a subset $V' \subseteq V$ with $|V'| \leq k$

The \textit{minimum $k$-median problem} consists in finding a $k$-median set $V'$ that minimizes

$$\sum_{u \in V} \min_{v \in V'} \text{dist}(u, v)$$

In the minimum \textit{metric} $k$-median problem (briefly, \textit{MMP}) the distance function is symmetric and satisfies the triangle inequality
Theorem

There is no approximation algorithm for the metric minimum k-median problem with approximation factor $\gamma < 1 + \frac{1}{e}$, unless $P = NP$.

Sketch of the proof:

It is based on an approximation factor preserving reduction from the minimum dominating set problem

Let $G = (V, E)$ be an undirected graph, a dominating set for $G$ is a subset $V' \subseteq V$ such that for each $u \in V \setminus V'$ there is a $v \in V'$ for which $\{u, v\} \in E$

The *minimum dominating set problem* consists in finding the minimum cardinality dominating set
Given an instance of the minimum dominating set problem, we define an instance of the minimum metric $k$-median problem with $G' = (V, E')$, $E' = V \times V$ and

$$\text{dist}(u, v) = \begin{cases} 
1 & \text{if } \{u, v\} \in E \\
2 & \text{otherwise.}
\end{cases}$$
Let us assume that there exists an approximation algorithm $\gamma$-$MMP$ with approximation factor $\gamma$ for $MMP$

Let us suppose that the size $k$ of an optimal dominating set is known

We devise an algorithm for the minimum dominating set

- Select a set of size $k$ by applying $\gamma$-$MMP$ with parameter $k$
- Remove the nodes in the graph corresponding to the chosen set and their neighbors
- Repeat until all the nodes are covered

$k = 2$
Let $\lambda$ be the number of iterations (the number of times that we apply $\gamma$-\textit{MMP})

At each iteration we selected $k$ nodes

We selected $k \cdot \lambda$ nodes

As $k$ is the value of the optimal solution, $\lambda$ is the approximation ratio of the algorithm for the minimum dominating set problem
We give an upper bound for $\lambda$:

After the first iteration,

- there are $k$ selected nodes, $d_1$ nodes covered \textit{directly} (with weight 1), $i_1$ nodes covered \textit{indirectly} (with weight 2), $k + d_1 + i_1 = |V| = n$
- The cost for MMP is $d_1 + 2i_1 \leq \gamma \text{OPT} \leq \gamma(n - k)$
- Therefore, $i_1 \leq (n - k)(\gamma - 1) \leq n(\gamma - 1)$

After $\lambda - 1$ iterations there are at most $n(\gamma - 1)^{\lambda - 1} = \eta$ uncovered nodes, for some $1 \leq \eta \leq n$, and then, $\lambda - 1 = \log(\gamma - 1) \frac{\eta}{n} \leq \log(\gamma - 1) \frac{1}{n} = \frac{\ln n}{\ln \frac{1}{\gamma - 1}}$

Cannot exists a $(c \ln n)$-approximation algorithm for the minimum dominating set for each $c < 1$, unless $P = NP$

Therefore, $\frac{1}{\ln \frac{1}{\gamma - 1}} \geq 1$ which implies $\frac{1}{\gamma - 1} \leq e$, and hence $\gamma \geq 1 + \frac{1}{e}$
Theorem

**MSP is at least as hard to approximate as the metric k-median problem.**

Corollary

There is no approximation algorithm for MSP with approximation factor \( \gamma < 1 + \frac{1}{e} \), unless \( P = NP \).

Theorem

For directed graphs, MSP does not belong to APX, unless \( P = NP \).
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
We define a local search algorithm as follows

- Given any initial solution $S_0$
- *Swap operation* of $t \leq |S|$ nodes:
  - remove $t$ nodes from $S$ and add $t$ nodes in $V \setminus S$ to $S$
- If any swap move yields a solution of lower cost the resulting solution is set to be the new current solution
- Repeat until from the current solution no swap operation decreases the cost
- The solution found represents a local optimum
We first analyze the case of $t = 1$: a swap is defined between two nodes $s \in S$ and $s' \in V \setminus S$ and consists in adding $s'$ and removing $s$

Let us define
\[
 f : (0, 1] \to \mathbb{R}, \quad f(\alpha) = \frac{2}{\alpha} \\
g : [0, \frac{1}{2}) \to \mathbb{R}, \quad g(\alpha) = \frac{12\alpha}{1 - 2\alpha} \\
h : [0, 1] \to \mathbb{R}
\]

\[
h(\alpha) = \begin{cases} 
 g(\alpha) & \text{if } \alpha = 0 \\
 \min\{f(\alpha), g(\alpha)\} & \text{if } \alpha \in (0, \frac{1}{2}) \\
 f(\alpha) & \text{if } \alpha \in \left[\frac{1}{2}, 1\right]
\end{cases}
\]
Theorem

The local search algorithm for MSP exhibits a locality gap of at most $5 + h(\alpha)$.

Maximum: $\approx 12.3$ for $\alpha \approx 0.274$ where $f(\alpha) = g(\alpha) \approx 7.3$
We can generalize for $t \geq 1$: the locality gap is $h'(\alpha)$, where

$$f' : (0, 1] \to \mathbb{R}, \quad f'(\alpha) = 1 + \frac{t+1}{t} \frac{1+2\alpha}{\alpha}$$

$$g' : [0, \frac{t}{t+1}] \to \mathbb{R}, \quad g'(\alpha) = \frac{(3+\alpha)t+2+\alpha}{(1-\alpha)t-\alpha}$$

$$h' : [0, 1] \to \mathbb{R}$$

$$h'(\alpha) = \begin{cases} 
    g'(\alpha) & \text{if } \alpha = 0 \\
    \min\{f'(\alpha), g'(\alpha)\} & \text{if } \alpha \in \left(0, \frac{t}{t+1}\right) \\
    f'(\alpha) & \text{if } \alpha \in \left[\frac{t}{t+1}, 1\right]
\end{cases}$$

Maximum: $\approx 8.67, 7.78$ and $7.05$
The algorithm can have a superpolynomial number of iterations.

We change the stopping condition: it finishes as soon as it finds a solution $S$ is such that every neighboring solution $S'$ of $S$ has $\text{cost}(S') > (1 - \epsilon)\text{cost}(S)$, for some $\epsilon > 0$.

The number of iterations is at most $\frac{\log(\frac{\text{cost}(S_0)}{\text{cost}(S^*)})}{\log(\frac{1}{1-\epsilon})}$, where $S_0$ is the initial solution.

**Corollary**

There exists an $\frac{1}{1-\epsilon} h'(\alpha)$-approximation algorithm MSP for any $\epsilon \in (0, 1)$. 
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
We implemented the local search algorithm in C++ (gcc).

We compared the solution found with the optimal one obtained by an IP formulation (GLPK solver).

\[
\begin{align*}
\min & \quad \sum_{v,s \in V} x_{vs} \cdot s_d(v) (d(v, s) + \alpha d(s, r)) \\
\text{s.t.} & \quad \sum_{s \in V} x_{vs} = 1 \quad \text{for each } v \in V \\
& \quad x_{vs} \leq y_s \quad \text{for each } v, s \in V \\
& \quad \sum_{s \in V} y_s \leq k, \\
& \quad y_r = 1 \\
& \quad y_s, x_{vs} \in \{0, 1\} \quad \text{for each } v, s \in V
\end{align*}
\]
Input instances

Types of graphs:
- **Random geometric graphs** (RGG) \( n \in \{100, 300, 1000\} \)
- **Barabasi-Albert graphs** (BA) \( n \in \{100, 300, 1000\} \)
- **OR Library** (PMED) \( 100 \leq n \leq 900 \)
- **Erdős-Rényi random graphs** (ER) \( n \in \{100, 150\} \)

Other parameters:
- The sink node is chosen uniformly at random
- \( \alpha \in \{0.0, 0.1, \ldots, 1\} \) (11 values)
- \( k \in \{1, \ldots, n\} \) (30 values with step \( \lfloor n/30 \rfloor \))
- \( s_d(v) \) uniformly at random in the interval \( [1, 10] \), independently for each \( v \in V \)
- \( \epsilon \) in \( \{0.005, 0.01, 0.1\} \)
- \( t = 1 \) (worst case for the algorithm’s approximation ratio)
Random geometric graphs – approximation ratio

- The ratio decreases with $\epsilon$, for $\epsilon = 0.005$ it is $< 1.108$
- When $k$ is small, the approximation ratio is reduced, for $k < 100$, it is $< 1.07$
- When $k$ is big, the approximation ratio is reduced, for $k > 250$, it is $< 1.05$

Figure: Random Geometric Graphs $n = 300$, $\alpha = 0.1$
Decreasing $\epsilon$ increases the number of iterations
The good values for small $k$ require up to 18 iterations
The good values for big $k$ require up to 2 iterations
For small values of $\epsilon$ the ratio is very small, $< 1.023$ for $\epsilon = 0.005$ and $< 1.042$ for $\epsilon = 0.01$.

Maximum value: 1.38, obtained when $\alpha = 0$
Random geometric graphs – number of iterations

The good values of the approximation ratio required more iterations if $\alpha$ is small.

When $\alpha$ approaches 1, then the usage of storage nodes does not significantly decrease the objective function and hence the first feasible solution already has a good approximation ratio.
We do not observe any significant difference with respect to the type of graph.

In these cases the approximation ratio is smaller than the previously reported ones.
### Computational time

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>n</th>
<th>k</th>
<th>Time per iteration (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGG</td>
<td>100</td>
<td>50</td>
<td>0.0121</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>0.3274</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>500</td>
<td>15.0086</td>
</tr>
<tr>
<td>BA</td>
<td>100</td>
<td>50</td>
<td>0.0122</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>0.3291</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>500</td>
<td>14.2448</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>n</th>
<th>k</th>
<th>Time per iteration (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMED</td>
<td>100</td>
<td>50</td>
<td>0.0116</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>150</td>
<td>0.3191</td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>450</td>
<td>10.9442</td>
</tr>
<tr>
<td>ER</td>
<td>100</td>
<td>50</td>
<td>0.0137</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>75</td>
<td>0.0392</td>
</tr>
</tbody>
</table>

**Table**: Average computational time required for each iteration when \( k = n/2 \)

The computational time of the iterations in the extreme cases, i.e. \( k = 1 \) or \( n = k \) is always < 0.0001
Outline

1. The Minimum $k$-Storage Problem
2. Polynomial-time exact algorithms
3. Hardness of approximation
4. Local search algorithm
5. Experimental analysis
6. Conclusions
Conclusions

- We studied the minimum $k$-storage problem from the theoretical and experimental viewpoints
- Directed graphs:
  - There exists a polynomial-time exact algorithm for trees
  - The problem is not in $APX$
- Undirected graph:
  - There exists a polynomial-time exact algorithm for bounded-treewidth graphs
  - The problem is not approximable within a factor of $1 + \frac{1}{e}$, unless $P = NP$
  - There exists a constant-factor polynomial-time approximation algorithm based on local search
  - This algorithm performs very well in practical scenarios

Thank you for your attention