Near Linear Time 5/3-Approximation Algorithms for Two-Level Power Assignment Problems

Benjamin Grimmer, Kan Qiao

Illinois Institute of Technology

August 11th, 2014
A Network Design Problem

Consider a network of ad hoc nodes that can operate at one of two power levels.

High power state allows a node to broadcast farther.

Problem
How do you choose which Ad Hoc nodes to assign high power to produce a connected network with minimum power-consumption?
Formal Problem Statement

Input is an directed graph $G = (V, E)$ and arc costs in $\{0, 1\}$. We assume $G$ is bidirected (arc $uv$ exists if and only if arc $vu$ exists, and they have the same cost).

**Symmetric Problem**
Assigning $S \subseteq V$ high power induces a digraph $H(S)$ with all zero-cost arcs and any one-cost arc between high power nodes.

**Asymmetric Problem**
Assigning $S \subseteq V$ high power induces a digraph $H(S)$ with all zero-cost arcs and any one-cost arc with a high power source endpoint.
A $\rho$-approximation is at most $\rho$ times larger than the optimal solution.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Author</th>
<th>Ratio</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>Nutov 2009</td>
<td>1.5</td>
<td>Large Polynomial</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Karim et al. 2014</td>
<td>$11/7 \approx 1.57$</td>
<td>$O(n^{10})$</td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Calinescu 2014</td>
<td>$1.61 + \epsilon$</td>
<td>$O(n^{k(\epsilon)})$</td>
</tr>
</tbody>
</table>

Table 1: Notable Previous Approximations, $n = |V|$ and $m = |E|$
## Faster Approximation Algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>Author</th>
<th>Ratio</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>Llyod et al. 2006</td>
<td>5/3</td>
<td>$O(nm\alpha(n))$</td>
</tr>
<tr>
<td>Symmetric</td>
<td></td>
<td><strong>5/3</strong></td>
<td><strong>$O(m\alpha(n))$</strong></td>
</tr>
<tr>
<td>Asymmetric</td>
<td>Calinescu 2014</td>
<td>7/4</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Asymmetric</td>
<td></td>
<td><strong>5/3</strong></td>
<td><strong>$O(m\alpha(n))$</strong></td>
</tr>
</tbody>
</table>

Table 2: Fast Approximations, Our results in bold, $n = |V|$ and $m = |E|$.

Note: $\alpha(n)$ is the inverse Ackermann function, which is practically constant.
Our Approach - Perfect Sets

We use $S \subseteq V$ to denote a partial solution.

**Definition**

$Q \subseteq V$ is **quasiperfect** with respect to $S$ if each node of $Q$ is in a different component of $H(S)$ and a single component of $H(S \cup Q)$.

**Definition**

A quasiperfect $Q$ is **perfect** with respect to $S$ the $Q$’s component in $H(S \cup Q)$ has no arcs leaving it.

**Lemma**

Any quasiperfect $Q$ can be augmented to become a perfect set $\text{Augment}(Q)$, such that $Q \subseteq \text{Augment}(Q)$. This can be computed by inspecting all edges incident to $\text{Augment}(Q)$ once.
Our Approach - Perfect Set Example

Figure 1: (a) Dotted boxes represent the initial connected components.
Our Approach - Perfect Sets Example

Figure 1: (a) Dotted boxes represent the initial connected components. (b) Perfect set (in gray) for both symmetric and asymmetric problems.
Our Approach - Perfect Set Example

Figure 1: (a) Dotted boxes represent the initial connected components. (b) Perfect set (in gray) for both symmetric and asymmetric problems. (c) Perfect set (in gray) for only the asymmetric problem.
Our Approach - Algorithm

1: Set $i = 0$; $S_0 = \emptyset$
2: \textbf{while} \exists Q \text{ perfect w.r.t. } S_i \text{ and } |Q| \geq 4 \text{ do}
3: \hspace{1em} S_{i+1} := S_i \cup Q; \ i := i + 1;
4: \hspace{-2em} \textbf{end while}
5: \textbf{while} \exists Q \text{ perfect w.r.t. } S_i \text{ and } |Q| = 3 \text{ do}
6: \hspace{1em} S_{i+1} := S_i \cup Q; \ i := i + 1;
7: \hspace{-2em} \textbf{end while}
8: \textbf{while} \exists Q \text{ perfect w.r.t. } S_i \text{ and } |Q| = 2 \text{ do}
9: \hspace{1em} S_{i+1} := S_i \cup Q; \ i := i + 1;
10: \hspace{-2em} \textbf{end while}
11: \textbf{return} \ S_i
Union Find Data Structure:

- Maintains disjoint groups for on a set of items.
- Looking up the group of an item takes $O(\alpha(n))$.
- Unioning two groups together takes $O(\alpha(n))$

We will use this to track the component of each vertex.
Definition
The restricted version of perfect sets for the symmetric problem will be called **symmetric perfect sets**.

Lemma
*For any symmetric perfect set Q, the subgraph with the vertices of Q and one-cost arcs between them is connected.*
Symmetric Implementation - Size Four or More

Fact

Any connected graph with at least four nodes either has a node of degree three or more, or contains a three-path.
Symmetric Implementation - Size Four or More

Fact

Any connected graph with at least four nodes either has a node of degree three or more, or contains a three-path.

1: for all $u \in V$ do
2: if $u$ adjacent to three or more other components in $H(S)$ then
3: $S_{i+1} := S_i \cup \text{Augment}(\{u\}); \ i := i + 1; \ \text{Union components of this set}$
4: end if
5: end for
Fact

Any connected graph with at least four nodes either has a node of degree three or more, or contains a three-path.

1: for all $u \in V$ do
2: if $u$ adjacent to three or more other components in $H(S)$ then
3: $S_{i+1} := S_i \cup \text{Augment} \{u\}; \ i := i + 1; \ \text{Union components of this set}$
4: end if
5: end for
6: for all $uv \in E$ do
7: if $|\text{Adjacent}[u] \cup \text{Adjacent}[v]| = 4$ then
8: $S_{i+1} := S_i \cup \text{Augment} \{u, v\}; \ i := i + 1; \ \text{Union components of this set}$
9: end if
10: end for
Symmetric Implementation - Algorithm

1: Set $i = 0; S_0 = \emptyset$
2: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| \geq 4$ do
3:    $S_{i+1} := S_i \cup Q; \ i := i + 1$
4: end while
5: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| = 3$ do
6:    $S_{i+1} := S_i \cup Q; \ i := i + 1$
7: end while
8: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| = 2$ do
9:    $S_{i+1} := S_i \cup Q; \ i := i + 1$
10: end while
11: return $S_i$
Asymmetric Implementation
Definition
At any point during our algorithm, we consider the **component graph** to be a graph where each component of $H(S)$ is a vertex, and components are adjacent if $G$ has an arc between them.

Definition
A quasiperfect set with vertices $q_1, q_2...q_k$ where $q_i$ has an arc to $\text{Comp}(q_{i+1})$ (and $q_k$ to $\text{Comp}(q_1)$) is called a **component cycle**.

Lemma
Every component cycle corresponds to a cycle in the component graph. Every perfect set contains a component cycle.
Asymmetric Implementation - Size Four or More Approach

1: while \( \exists Q \) symmetric perfect w.r.t. \( S_i \) and \( |Q| \geq 4 \) do
2: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
3: end while
4: while \( \exists Q \) perfect w.r.t. \( S_i \) and \( \exists C \subseteq Q \) component cycle and \( |C| \geq 4 \) do
5: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
6: end while
7: while \( \exists Q \) perfect w.r.t. \( S_i \), \( |Q| \geq 4 \) and \( \exists C \subseteq Q \) component cycle and \( |C| = 3 \) do
8: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
9: end while
10: while \( \exists Q \) perfect w.r.t. \( S_i \), \( |Q| \geq 4 \) and \( \exists C \subseteq Q \) component cycle and \( |C| = 2 \) do
11: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
12: end while
Asymmetric Implementation - Size Four or More Approach

1. while ∃Q symmetric perfect w.r.t. $S_i$ and $|Q| \geq 4$ do
2.    $S_{i+1} := S_i \cup Q; \ i := i + 1;$
3. end while
4. while ∃Q perfect w.r.t. $S_i$ and ∃C ⊆ Q component cycle and $|C| \geq 4$ do
5.    $S_{i+1} := S_i \cup Q; \ i := i + 1;$
6. end while
7. while ∃Q perfect w.r.t. $S_i$, $|Q| \geq 4$ and ∃C ⊆ Q component cycle and $|C| = 3$ do
8.    $S_{i+1} := S_i \cup Q; \ i := i + 1;$
9. end while
10. while ∃Q perfect w.r.t. $S_i$, $|Q| \geq 4$ and ∃C ⊆ Q component cycle and $|C| = 2$ do
11.    $S_{i+1} := S_i \cup Q; \ i := i + 1;$
12. end while
Asymmetric Implementation - Cycle Detection

We will use a depth first search on the component graph to find component cycles.

Figure 2: (a) A Depth First Search (DFS) with edges to be processed.
Asymmetric Implementation - Cycle Detection

We will use a depth first search on the component graph to find component cycles.

Figure 2: (a) A Depth First Search (DFS) with edges to be processed. (b) The three types of cycles contracted during the DFS.
Asymmetric Implementation - Cycle Branches

Definition
An arc going from a component cycle (or any quasiperfect set) to a component outside of the set is a **branch**.

Figure 3: (a) A component cycle with a branch to a component in the path. (b) A component cycle with a branch to an undiscovered component (dashed).
Asymmetric Implementation - Size Four Cycle Contraction

Figure 4: When a component cycle is found, any branch to the path can be removed as shown by the Expand operation. After Expand runs, all remaining tails to undiscovered components are handled by Augment. The final perfect set can be contracted, combining all $E_{new}$. 
Asymmetric Implementation - Size Four or More Approach

1: while \( \exists Q \) symmetric perfect w.r.t. \( S_i \) and \( |Q| \geq 4 \) do
2: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
3: end while
4: while \( \exists Q \) perfect w.r.t. \( S_i \) and \( \exists C \subseteq Q \) component cycle and \( |C| \geq 4 \) do
5: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
6: end while
7: while \( \exists Q \) perfect w.r.t. \( S_i \), \( |Q| \geq 4 \) and \( \exists C \subseteq Q \) component cycle and \( |C| = 3 \) do
8: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
9: end while
10: while \( \exists Q \) perfect w.r.t. \( S_i \), \( |Q| \geq 4 \) and \( \exists C \subseteq Q \) component cycle and \( |C| = 2 \) do
11: \( S_{i+1} := S_i \cup Q; \ i := i + 1; \)
12: end while
Asymmetric Implementation - Algorithm

1: Set $i = 0; S_0 = \emptyset$
2: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| \geq 4$ do
3:     $S_{i+1} := S_i \cup Q; i := i + 1$
4: end while
5: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| = 3$ do
6:     $S_{i+1} := S_i \cup Q; i := i + 1$
7: end while
8: while $\exists Q$ perfect w.r.t. $S_i$ and $|Q| = 2$ do
9:     $S_{i+1} := S_i \cup Q; i := i + 1$
10: end while
11: return $S_i$
Conclusion

- We presented a greedy algorithm for two Power Assignment problems with a 5/3-approximation ratio.
- In the Symmetric Case, we gave a $O(m\alpha(n))$ implemention.
- We outlined how to extend this for the Asymmetric Problem using a modified depth first search.
Conclusion

- We presented a greedy algorithm for two Power Assignment problems with a $5/3$-approximation ratio.
- In the Symmetric Case, we gave a $O(m\alpha(n))$ implementation.
- We outlined how to extend this for the Asymmetric Problem using a modified depth first search.

Thank You.
Tightness of Approximation Bound