Approximating the Held-Karp Bound for Metric TSP in Nearly Linear Time

Chandra Chekuri
Kent Quanrud
UIUC
Metric TSP

**version 1**

**Input:** clique $K_n$, costs $c : V \times V \rightarrow \mathbb{R}_{\geq 0}$ forming a metric

**Objective:** min-cost Hamiltonian cycle in $(K_n, c)$

**version 2**

**Input:** graph $G = (V, E)$ nonnegative costs $c \in \mathbb{R}^E$

**Objective:** min-cost tour in $(G, c)$

---

Metric completion
Metric TSP

**version 1**

**Input:** clique $K_n$, costs $c : V \times V \to \mathbb{R}_{\geq 0}$ forming a metric

$(K_n, c)$ is dense

(size $\Omega(n^2)$)

**Objective:** min-cost Hamiltonian cycle in $(K_n, c)$

**version 2**

**Input:** graph $G = (V, E)$, nonnegative costs $c \in \mathbb{R}^E$

$(G, c)$ is sparse (size $O(m)$)

**Objective:** min-cost tour in $(G, c)$

**metric completion**
(Metric) Subtour Elimination

Input: clique \( K_n = (V, E) \), metric \( c : E \rightarrow \mathbb{R}_{\geq 0} \)

Objective: \[
\min \sum_{e \in E} c_e x_e \text{ over } x \in \mathbb{R}^E
\]

Degree constraints:
\[
\sum_{e \in \mathcal{C}(v)} x_e = 2 \text{ for all vertices } v;
\]

eliminates subtours:
\[
\sum_{e \in \mathcal{C}(U)} x_e \geq 2 \text{ for all sets } U \neq \emptyset, V;
\]

and \( 0 \leq x \leq 1 \)

equivalent to Held-Karp bound
Related work (abbrev.)

- important for variants of Christofides algorithm
- solvable by ellipsoid method

\[ \tilde{O}(n^4/\epsilon^2) \] 
\[ \tilde{O}(m^2/\epsilon^2) \]

*Plotkin, Shmoys, Tardos [1995]*

*Garg and Khandekar [2004]*

metric completion
Metric TSP

**version 1**

**Input:** clique $K_n$, costs $c : V \times V \rightarrow \mathbb{R}_{\geq 0}$ forming a metric

**Objective:**
min-cost Hamiltonian cycle in $(K_n, c)$

**version 2**

**Input:** graph $G = (V, E)$, nonnegative costs $c \in \mathbb{R}^E$

**Objective:**
min-cost tour in $(G, c)$
Metric TSP

version 1

Input: clique $K_n$, costs $c : V \times V \rightarrow \mathbb{R}_{\geq 0}$ forming a metric

Objective: min-cost Hamiltonian cycle in $(K_n, c)$

Question: $(1 - \varepsilon)$-APX for Held-Karp bound in nearly-linear $\tilde{O}(m/\text{poly}(\varepsilon))$ time?

version 2

Input: graph $G = (V, E)$ nonnegative costs $c \in \mathbb{R}^E$

Objective: min-cost tour in $(G, c)$
Metric TSP

**version 1**

**Input:** clique $K_n$, costs $c : V \times V \to \mathbb{R}_{\geq 0}$ forming a metric

**Objective:** min-cost Hamiltonian cycle in $(K_n, c)$

**Yes:** $(1 - \epsilon)$-APX for Held-Karp in $\tilde{O}(m/\epsilon^2)$

**version 2**

**Input:** graph $G = (V, E)$, nonnegative costs $c \in \mathbb{R}^E$

**Objective:** min-cost tour in $(G, c)$
Held-Karp for Metric TSP

Packing cuts

Dynamic min cuts (and updates)
Held-Karp for Metric TSP

Part 2
Packing cuts

Dynamic min cuts (and updates)

Part 3
2-edge connected spanning subgraph (2ECSS)

Input: graph $G = (V, E)$ with cuts $\mathcal{C}$
nonnegative edge costs $c \in \mathbb{R}_{\geq 0}^E$

Objective: \[
\min \sum_{e \in E} c_e y_e \quad \text{over} \quad y \in \mathbb{R}^E
\]
\[\sum_{e \in \mathcal{C}} y_e \geq 2 \quad \text{for all cuts} \quad C \in \mathcal{C}\]
and $y \geq 0^E$
equivalent to subtour elimination on metric completion of $(G, c)$
Dual of 2ECSS (2ECSSD)

Input: graph $G = (V, E)$ with cuts $\mathcal{C}$, costs $c \in \mathbb{R}^E_{\geq 0}$

Objective: \[
\max \ 2 \sum_{c \in \mathcal{C}} x_C \text{ over } x \in \mathbb{R}^\mathcal{C} \\
\text{s.t. } \sum_{C \ni e} x_C \leq c_e \text{ for all edges } e, \\
\text{and } x \geq \emptyset^\mathcal{C}
\]
Dual of 2ECSS (2ECSSD)

**Input:** graph $G = (V, E)$ with cuts $\mathcal{C}$, costs $c \in \mathbb{R}^E_{\geq 0}$

**Objective:**

$$\text{max } 2 \sum_{c \in \mathcal{C}} x_C \text{ over } x \in \mathbb{R}^\mathcal{C}$$

s.t. $\sum_{\mathcal{C} \ni e} x_C \leq c_e$ for all edges $e$,

and $x \geq 0^\mathcal{C}$
Held-Karp for Metric TSP

Dynamic min cuts (and updates)

Part 2

Packing cuts

Part 1
Multiplicative weight updates

Cut packings $\max \sum_{C \in C} x_C \text{ over } x \in \mathbb{R}^C$

$\text{s.t. } \sum_{C \ni e} x_C \leq c_e \quad e \in E$

$x \geq 0^C$

Knapsack problems

0. Initialize edge weights $w \leftarrow 1/c$

1. Solve relaxation

$\max \sum_{C} x_C \text{ s.t. } x \geq 0,$

$s.t. \sum_{c, e : e \in C} w_e x_c \leq \sum_{e} w_e c_e$

3. Output convex combination of relaxed solutions

$\tilde{O}(m/\epsilon^2)$ iterations

for each $e \in E$

$w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} x_C}{\max_h \sum_{C \ni h} x_C/c_h} \right) w_e$
**Multiplicative weight updates**

1. Solve relaxation...
2. For each $e \in E$
   \[ w_e \leftarrow \exp \left( \frac{\epsilon \sum_{c \ni e} x_C c_e}{\max_h \sum_{c \ni h} x_C c_h} \right) w_e \]

0. Initialize edge weights $w \leftarrow 1/c$

Solve relaxation:
\[
\max \sum_{C} x_C \quad \text{s.t.} \quad x \geq 0, \quad \sum_{c,e : e \in c} w_e x_c \leq \sum_{e} w_e c_e \quad \tilde{O}(m/\epsilon^2) \text{ iterations}
\]
**Multiplicative weight updates**

1. Solve relaxation...
   - **a** \( C \leftarrow \text{min-cut}(G, w) \) \( \tilde{O}(m) \) time [Karger]
   - **b** \( x \leftarrow \frac{\langle w, c \rangle}{\sum_{e \in T} w_e} e_T \)

2. For each \( e \in E \)
   \[
   w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} x_C c_e}{\max_h \sum_{C \ni h} x_C c_h} \right) w_e
   \]

0. Initialize edge weights \( w \leftarrow 1/c \)

Solve relaxation

\[
\max \sum_C x_C \text{ s.t. } x \geq 0,
\]

\[
\sum_{c, e : e \in c} w_e x_c \leq \sum_e w_e c_e
\]

\( \tilde{O}(m/\epsilon^2) \) iterations
**Multiplicative weight updates**

cut packings  \[\xrightarrow{\text{MWU}}\]  min-weight cuts

2. **for each** $e \in E$
   \[w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} \frac{x_C}{c_e}}{\max_h \sum_{C \ni h} \frac{x_C}{c_h}} \right) w_e\]

0. **initialize edge weights** $w \leftarrow 1/c$

1. a. $C \leftarrow \text{min-cut}(G, w)$
   \[\tilde{O}(m)\text{ time [Karger]}\]

1. b. $x \leftarrow \frac{\langle w, c \rangle}{\sum_{e \in T} w_e} e_T$
   \[\tilde{O}(m/\epsilon^2)\text{ iterations}\]

2. **for each** $e \in E$
   \[w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} \frac{x_C}{c_e}}{\max_h \sum_{C \ni h} \frac{x_C}{c_h}} \right) w_e\]
Multiplicative weight updates

**cut packings** \[ \quad \text{for each } e \in E \]
\[ w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} x_C}{\max_h \sum_{C \ni h} x_C} \right) w_e \]
\[ \Rightarrow \text{for each } e \in C, \]
\[ w_e \leftarrow \exp \left( \frac{\epsilon \min_{h \in C} c_h}{c_e} \right) w_e \]

**MWU** \[ \quad \text{initialize edge weights } w \leftarrow 1/c \]

**min-weight cuts**

1. \[ a \] \[ C \leftarrow \min\text{-cut}(G, w) \]
\[ \tilde{O}(m) \text{ time [Karger]} \]

2. \[ b \] \[ x \leftarrow \frac{\langle w, c \rangle}{\sum_{e \in T} w_e} e_T \]
\[ \tilde{O}(m/\epsilon^2) \text{ iterations} \]

3. \[ 2 \] \[ \text{for each } e \in E \]
\[ w_e \leftarrow \exp \left( \frac{\epsilon \sum_{C \ni e} x_C}{\max_h \sum_{C \ni h} x_C} \right) w_e \]
Multiplicative weight updates

cut packings  \[\xrightarrow{\text{MWU}}\]  min-weight cuts

0. Initialize edge weights: \( w \leftarrow 1/c \)

1a. \( C \leftarrow \text{min-cut}(G, w) \)

\( \tilde{O}(m) \text{ time [Karger]} \)

1b. \( x \leftarrow \frac{\langle w, c \rangle}{\sum_{e \in T} w_e} e_T \)

\( \tilde{O}(m/\epsilon^2) \text{ iterations} \)

2. For each \( e \in C \), \( w_e \leftarrow \exp \left( \frac{\epsilon \min_{h \in C} c_h}{c_e} \right) w_e \)
**Multiplicative weight updates**

Cut packings  \( \rightarrow \) MWU  \( \rightarrow \) Min-weight cuts

2 bottlenecks

0. Initialize edge weights: \( w \leftarrow 1/c \)

1. \( a \)
   - Compute min-cut: \( C \leftarrow \text{min-cut}(G, w) \)
   - \( \tilde{O}(m) \) time [Karger]

2. \( b \)
   - \( x \leftarrow \frac{\langle w, c \rangle}{\sum_{e \in T} w_e} e_T \)
   - \( \tilde{O}(m/\epsilon^2) \) iterations

3. For \( e \in C, \ w_e \leftarrow \exp \left( \frac{\epsilon \min_{h \in C} c_h}{c_e} \right) w_e \)
Multiplicative weight updates

cut packings ➞ MWU ➞ min-weight cuts

\[ C \leftarrow \text{min-cut}(G, w) \]
\[ \tilde{O}(m) \quad \text{per min cut} \]
\[ \times \tilde{O}(m/\epsilon^2) \quad \text{iterations} \]
\[ \times \tilde{O}(m^2/\epsilon^2) \quad \text{running time} \]

\[ \text{initialize edge weights } w \leftarrow 1/c \]
\[ \tilde{O}(m) \text{ time [Karger]} \]

\[ \text{for } e \in C, \ w_e \leftarrow \ldots \]
\[ \Omega(m) \quad \text{edge updates per cut} \]
\[ \times \tilde{O}(m/\epsilon^2) \quad \text{iterations} \]
\[ \times \tilde{O}(m^2/\epsilon^2) \quad \text{running time} \]
<table>
<thead>
<tr>
<th>What we have</th>
<th>What we need</th>
<th>Additional challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tilde{O}(m)) per min cut</td>
<td>(\tilde{O}(1)) amort. per ((1 + \epsilon))-apx min-cut</td>
<td>no suitable dynamic data structures</td>
</tr>
<tr>
<td>(\Omega(m)) edges per cut</td>
<td>(\tilde{O}(1)) amortized time per cut</td>
<td>min-cut varies dramatically between iterations</td>
</tr>
</tbody>
</table>
Held-Karp for Metric TSP

Packing cuts

Dynamic min cuts (and updates)

Part 2

Part 1
Held-Karp for Metric TSP

Packing cuts

Dynamic min cuts (and updates)

Part 1

Part 2

Part 3
Karger’s $\tilde{O}(m)$ min-cut algorithm

1. Randomly contract edge. Repeat.
Karger’s $\tilde{O}(m)$ min-cut algorithm

1. Randomly contract edge. Repeat.

1. Pack spanning trees

2. Randomly sample $O(\log n)$ trees

3. Search each sampled tree for min-cut induced by 1 or 2 edges by dynamic programming
Tree packings and network strength

**Input:** graph $G = (V, E)$ w/ spanning trees $\mathcal{T}$ and positive edge capacities $c \in \mathbb{R}^E$

**Objective:**

$$\max \sum_{T \in \mathcal{T}} x_T \text{ over } x \in \mathbb{R}^\mathcal{T}$$

subject to

$$\sum_{T \ni e} x_T \leq c_e \text{ for each edge } e$$

$$x \geq 0^\mathcal{T}$$

$(1 - \epsilon)$-apx tree packings in $\tilde{O}(m)$ time via either:

- sparsification [Karger 00]
- MWU [CQ SODA17]
Tree packings and network strength

Input: graph $G = (V, E)$ w/ spanning trees $\mathcal{T}$
and positive edge capacities $c \in \mathbb{R}^E$

Objective: $\max \sum_{T \in \mathcal{T}} x_t$ over $x \in \mathbb{R}^\mathcal{T}$

s.t. $\sum_{T \ni e} x_T \leq c_e$ for each edge $e$

$x \geq 0^\mathcal{T}$
Undirected graph w/ min cut $\kappa$ has a tree packing of value $\geq \kappa/2$.

Tight for cycle
- min cut = 2
- tree-packing = 1
Undirected graph w/ min cut $\kappa$ has a tree packing of value $\geq \kappa/2$

let $P$ be a $(1 - \epsilon)$-apx max tree packing
$C$ be a $(1 + \epsilon)$-apx min cut

each edge $e \in C$ is in a tree $T \in P$

 avg # $C$-edges per tree
$$\frac{|C|}{|P|} \leq \frac{(1 + \epsilon)\kappa}{(1 - \epsilon)\kappa/2} = 2 + O(\epsilon)$$

Markov $\Rightarrow$ const. fraction of trees have $< 3$ $C$-edges
Karger’s $\tilde{O}(m)$ min-cut algorithm

1. Randomly contract edge. Repeat.

1. Pack spanning trees

2. Randomly sample $O(\log n)$ trees

3. Search each sampled tree for min-cut induced by 1 or 2 edges by dynamic programming
Fix a rooted tree $T \Rightarrow$ poset on $V$

“$u < v$” means $u$ is a descendant of $v$

“$v \parallel w$” means $v$ and $w$ are incomparable

“$D(x)$” means all descendants of $x$
check value in $G$ of each cut induced by $\leq 2$ edges in $T$

1. 1-cut

$C(D(s))$

2. Incomparable 2-cut

$C(D(s) \cup D(t))$

(where $s \parallel t$)

3. Nested 2-cut

$C(D(t) \setminus D(s))$

(where $s \leq t$)

($C(S) = $ edges cut by $S$)
1-cut

$$\overline{w}(\mathcal{C}(D(s))) = \sum_{v \in D(s)} \sum_{e \in \mathcal{C}(v)} w(e)$$

- weight of cut
- weight of edges contained in $D(s)$

Tree sums over weighted degrees
$O(m)$ time total over all $s$

Tree sums over weights at lca’s
$\tilde{O}(m)$ time total over all $s$

build up sums in dynamic trees and read off weights
2-cuts are a little more complicated...

**Link-Cut trees**

- (a) add along $v \rightarrow$ root path
- (b) get min over $v \rightarrow$ root path

| 1. init each $v$ to $\overline{C}(D(v))$
| 2. add $\infty$ to all $v > s$
| 3. for each $e = (s, v)$
| a. subtract $2w(e)$ from all $u \geq v$
| 4. for each $e = (s, v)$
| a. find min value over all $u \geq v$

**nested 2-cut with a path to a leaf**

\[
\overline{C}(D(t) \setminus D(s_i), D(s_i)) = \overline{C}(D(t) \setminus D(s_i)) + 2\overline{C}(D(t) \setminus D(s_i), D(s_i))
\]

1. process $s_1$ like a leaf
2. for $i = 2, \ldots, \ell$
   a. keep aggr. values from $s_{i-1}$
   b. process edges incident to $s_i$ like a leaf
3. return best min over all $i$

**nested 2-cut with a leaf**

\[
\overline{C}(D(t) - s) = \overline{C}(D(t)) - \overline{C}(s) + 2\overline{C}(D(t) - s, s)
\]

1. init each $v$ to $\overline{C}(D(v))$
2. for each $e = (s, v)$
   a. add $2w(e)$ to all $u \geq v$
3. find min value over all $u \geq v$

**induction step**

1. process each path to a leaf
2. contract each path into its parent
3. recurse
2-cuts are a little more complicated...

reduces to dynamic programming with dynamic trees
Incremental setting

- edge weights incremented online (adversarially)
- need to maintain $(1 + \epsilon)$-apx min cut
Incremental Karger’s algorithm

- Initially: $\lambda \leftarrow$ initial value of min-cut, pack and sample $\log n$ spanning trees.

- When we need an apx min-cut:
  - Continue Karger’s search until we find a cut of value $\leq (1 + \varepsilon)\lambda$, output and pause the search.
  - If good cut not found, then re-pack/sample trees.

- When we increment an edge weight:
  - Incorporate into tree sums w/ dynamic trees.
<table>
<thead>
<tr>
<th>what we have</th>
<th>what we need</th>
<th>what we get</th>
</tr>
</thead>
<tbody>
<tr>
<td>From $\tilde{O}(m^2/\epsilon^2)$ to $\tilde{O}(m/\epsilon^2)$</td>
<td>min-cut oracle</td>
<td>weight update</td>
</tr>
<tr>
<td>$\tilde{O}(m)$ per min cut</td>
<td>$\tilde{O}(1)$ amort. per $(1 + \epsilon)$-apx min-cut</td>
<td>$\tilde{O}(m)$ edges per cut</td>
</tr>
<tr>
<td>$\tilde{O}(m/\epsilon^2)$ total time</td>
<td>$\tilde{O}(m/\epsilon^2)$ total time</td>
<td>$\tilde{O}(m/\epsilon^2)$ total time</td>
</tr>
<tr>
<td>+ $\tilde{O}(1)$ per min cut</td>
<td>+ $\tilde{O}(1)$ per min cut</td>
<td>+ $\tilde{O}(1)$ per min cut</td>
</tr>
<tr>
<td>+ $\tilde{O}(1)$ per edge inc</td>
<td>+ $\tilde{O}(1)$ per edge inc</td>
<td>+ $\tilde{O}(1)$ per edge inc</td>
</tr>
</tbody>
</table>
Updating edge weights along cuts

- need to update weights of all edges in a cut
- we know how to update fixed sets efficiently
  \[\text{[Young '14, Chekuri-Q SODA17]}\]
- problem: cuts vary dramatically between iterations
- key point: all cuts are induced by trees
Euler tour converts subtrees to intervals

Range tree decomposes each side of a 1,2-cut to \( \log n \)
“canonical subtrees”

this decomposes each 1,2-cut to \( \log^2 n \)
“canonical cuts” between canonical subtrees
Updating edge weights along cuts

- need to update weights of all edges in a cut
- we know how to update fixed sets efficiently
  \[ \text{[Young '14, Chekuri-Q SODA17]} \]
- problem: cuts vary dramatically between iterations
- key point: all cuts are induced by trees
- "canonical cuts" with total size \( \~O(m) \)

\[ \star + \star \Rightarrow \log^2 n \text{ efficient updates on fixed sets} \]
<table>
<thead>
<tr>
<th>What we have</th>
<th>What we need</th>
<th>What we get</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tilde{O}(m) ) per min cut</td>
<td>( \tilde{O}(1) ) amort. per ((1 + \epsilon))-apx min-cut</td>
<td>( \tilde{O}(m/\epsilon^2) ) total time + ( \tilde{O}(1) ) per min cut + ( \tilde{O}(1) ) per edge inc</td>
</tr>
<tr>
<td>( \Omega(m) ) edges per cut</td>
<td>( \tilde{O}(1) ) amortized time per cut</td>
<td>( \tilde{O}(m/\epsilon^2) ) init time + ( \tilde{O}(1) ) per min cut + ( \tilde{O}(m/\epsilon^2) ) edge increments</td>
</tr>
</tbody>
</table>
Held-Karp for Metric TSP

Packing cuts

Dynamic min cuts (and updates)

Part 1

Part 2

Part 3
The main result. In this paper we obtain a near-linear running time for a \((1+\epsilon)\)-approximation, substantially improving the best previously known running time bound.

**Theorem 1.1.** Let \(G = (V,E)\) be an undirected graph with \(|E| = m\) edges and \(|V| = n\) vertices, and positive edge weights \(c : E \rightarrow \mathbb{R}_{>0}\). For any fixed \(\epsilon > 0\), there exists a randomized algorithm that computes a \((1+\epsilon)\)-approximation to the Held-Karp lower bound for the Metric-TSP instance on \((G,c)\) in \(O(m \log^4 n/\epsilon^2)\) time. The algorithm succeeds with high probability.

The algorithm in the preceding theorem can be modified to return a \((1+\epsilon)\)-approximate solution to the 2ECSS LP within the same asymptotic time bound. For fixed \(\epsilon\), the running time we achieve is asymptotically faster than the time to compute or even write down the metric completion of \((G,c)\). Our algorithm can be applied low-dimensional geometric point sets to obtain a running-time that is near-linearly in the number of points.

In typical approximation algorithms that rely on mathematical programming relaxations, the bottleneck for the running time is solving the relaxation. Surprisingly, for algorithms solving Metric-TSP via the Held-Karp bound, the bottleneck is no longer solving the relaxation (albeit we only find a \((1+\epsilon)\)-approximation and do not guarantee a basic feasible solution). We mention that the recent approaches towards the \(4/3\) conjecture for Metric-TSP are based on variations of the classical Christofides heuristic (see [Vygen, 2012]). The starting point is a near-optimal feasible solution \(x\) to the 2ECSS LP on \((G,c)\). Using a well-known fact that a scaled version of \(x\) lies in the spanning tree polytope of \(G\), one generates one or more (random) spanning trees \(T\) of \(G\). The tree \(T\) is then augmented to a tour via a min-cost matching \(M\) on its odd degree nodes. Genova and Williamson [2017] recently evaluated some of these Best-of-Many Christofides’ algorithms and demonstrated their effectiveness. A key step in this scheme, apart from solving the LP, is to decompose a given
Held-Karp for Metric TSP

Part 2

Packing cuts

Dynamic min cuts (and updates)

Part 1

Part 3

Recent work

Fast implementation of Christofides’ algo
Christofides’ heuristic [1976] 
(Recent work)

- simple (& best) 3/2-approximation for metric TSP
- bottlenecks include all-pairs shortest paths, min-cost perfect matching on dense graph

\[(1 + \epsilon)\text{-apx to 2ECSS} \Rightarrow\]

\[\left(1 + \epsilon\right)^\frac{3}{2}\text{-apx in } \tilde{O}(n^{1.5}/\epsilon^3) \text{ time}\]

\[\Rightarrow \tilde{O}(m/\epsilon^2 + n^{1.5}/\epsilon^3) \text{ time total}\]
Thanks!