From theory to practice in k-OPT heuristic for Travelling Salesman Problem

Łukasz Kowalik

(joint work with Marek Cygan, Arkadiusz Socała and Kamil Żyła)
Traveling Salesman Problem (TSP)

Input
complete undirected graph $G = (V, E)$ and
a weight function $w : E \to \mathbb{N}$.

Problem
Find a tour (Hamiltonian cycle) of minimum weight.
Traveling Salesman Problem (TSP)

**Input**
complete undirected graph $G = (V, E)$ and
a weight function $w : E \rightarrow \mathbb{N}$.

**Problem**
Find a tour (Hamiltonian cycle) of minimum weight.
The shortest tour catching all San Francisco pokemons
(from http://www.math.uwaterloo.ca/tsp/poke/)

 Łukasz Kowalik
From theory to practice in k-OPT heuristic for TSP
Solving TSP

- Problem is NP-hard
- Best exact algorithm in time $2^n n^{O(1)}$.
- No approximation possible in general (unless P=NP)
- Some nice approximation algorithm under additional assumptions
  - $w$ is a metric: 1.5-approximation (Christofides 1976)
  - Euclidean metric: a PTAS (Arora 1996)
  - Graphic metric: 1.4-approximation (Sebo, Vygen, 2012)
- In practice: people use heuristics.
1. $H_0 := \text{arbitrary Hamiltonian cycle.}$

2. As long as possible, get a better cycle $H_i$ by means of the $k$-move operation.
For a tour $H$, a $k$-move is defined by a pair $(E^-, E^+)$ such that

- $|E^-| = |E^+| = k$ and
- $H' = H \setminus E^- \cup E^+$ is a Hamiltonian cycle.

Example for $k = 3$:

$k$-move is improving when $w(H') < w(H)$. 
For a tour $H$, a **$k$-move** is defined by a pair $(E^-, E^+)$ such that

- $|E^-| = |E^+| = k$ and
- $H' = H \setminus E^- \cup E^+$ is a Hamiltonian cycle.

Example for $k = 3$:

$k$-move is **improving** when $w(H') < w(H)$. 

---

Łukasz Kowalik

From theory to practice in $k$-OPT heuristic for TSP
For a tour $H$, a $k$-move is defined by a pair $(E^-, E^+)$ such that

1. $|E^-| = |E^+| = k$ and
2. $H' = H \setminus E^- \cup E^+$ is a Hamiltonian cycle.

Example for $k = 3$:

$k$-move is **improving** when $w(H') < w(H)$. 

**Łukasz Kowalik**

From theory to practice in $k$-OPT heuristic for TSP
**k-move**

For a tour $H$, a *k-move* is defined by a pair $(E^-, E^+)$ such that

1. $|E^-| = |E^+| = k$ and
2. $H' = H \setminus E^- \cup E^+$ is a Hamiltonian cycle.

Example for $k = 3$:

$k$-move is **improving** when $w(H') < w(H)$. 
For a tour $H$, a $k$-move is defined by a pair $(E^-, E^+)$ such that

- $|E^-| = |E^+| = k$ and
- $H' = H \setminus E^- \cup E^+$ is a Hamiltonian cycle.

Example for $k = 3$:

$k$-move is **improving** when $w(H') < w(H)$. 
**Practice**
An implementation of a variant, called Lin-Kernighan heuristic solves 80K-vertex instances optimally (Hellsgaun ’09).

**Theory**
Interesting results (lower, upper bounds) on

- quality of local optima (e.g. Chandra et al, SICOMP’99),
- number of steps needed to find local optimum (e.g., Johnson et al, JCSS’88),
- smoothed analysis of 2-opt (e.g. Künnemann and B. Manthey, ICALP’15).
Today’s question

How fast can we perform a **single step**, i.e., How fast can we find an improving $k$-move?
Today’s question

**k-opt Optimization**

**Input:** symmetric function \( w : V^2 \to \mathbb{N} \), a Hamiltonian cycle \( H \)

**Output:** a \( k \)-move that maximizes improvement over \( H \).

**k-opt Detection**

**Output:** Is there a \( k \)-move improving over \( H \)?

**Upper bounds**

- \( O(n^k) \) exhaustive search,

**Lower bounds**

- \( W[1]\)-hard [Marx ’08]
- no \( n^{o(k/\log k)} \) algorithm under ETH [Guo et al. ’13]
- no \( o(n^2) \) algorithm for \( k = 2 \) (folklore),

Łukasz Kowalik  From theory to practice in k-OPT heuristic for TSP
Today’s question

**k-opt Optimization**

**Input:** symmetric function $w : V^2 \rightarrow \mathbb{N}$, a Hamiltonian cycle $H$

**Output:** a $k$-move that maximizes improvement over $H$.

**k-opt Detection**

**Output:** Is there a $k$-move improving over $H$?

**Upper bounds**
- $O(n^k)$ exhaustive search,
- $O(n^{2k/3} + 1)$ time, $O(n)$ additional space [de Berg, Buchin, Jansen, Woeginger ’16]

**Lower bounds**
- no $n^{o(k/\log k)}$ algorithm under ETH [Guo et al. ’13]
- no $o(n^2)$ algorithm for $k = 2$ (folklore),
- if $o(n^{2.99})$ algorithm for $k = 3$, then APSP in time $o(n^{2.99})$ [de Berg et al].

Łukasz Kowalik
From theory to practice in k-OPT heuristic for TSP
Part I: Theory
(joint work with Marek Cygan and Arkadiusz Socała)
Our results

Theorem

For every fixed integer $k$, $k$-opt Optimization can be solved in time $O(n^{(1/4+\epsilon_k)k})$ and space $O(n^{(1/8+\epsilon_k)k})$, where $\lim_{k \to \infty} \epsilon_k = 0$. 
Our results

Theorem

For every fixed integer \( k \), \( k \)-OPT Optimization can be solved in time \( O(n^{(1/4+\epsilon_k)k}) \) and space \( O(n^{(1/8+\epsilon_k)k}) \), where \( \lim_{k \to \infty} \epsilon_k = 0 \).

Values of \( \epsilon_k \) (computed by a program)

\[
\begin{array}{ccccccc}
  k & 3 & 4 & 5 & 6 & 7 & 8 \\
  \text{de Berg et al.} & O(n^3) & O(n^3) & O(n^4) & O(n^5) & O(n^5) & O(n^6) \\
  \text{our algorithm} & O(n^{3.4}) & O(n^4) & O(n^{4.25}) & O(n^{4\frac{2}{3}}) & \\
\end{array}
\]
Our results

Theorem

For every fixed integer $k$, $k$-opt Optimization can be solved in time $O(n^{(1/4+\epsilon_k)k})$ and space $O(n^{(1/8+\epsilon_k)k})$, where $\lim_{k \to \infty} \epsilon_k = 0$.

Values of $\epsilon_k$ (computed by a program)

<table>
<thead>
<tr>
<th>$k$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>de Berg et al.</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(n^4)$</td>
<td>$O(n^5)$</td>
<td>$O(n^5)$</td>
<td>$O(n^6)$</td>
</tr>
<tr>
<td>our algorithm</td>
<td>$O(n^{3.4})$</td>
<td>$O(n^4)$</td>
<td>$O(n^{4.25})$</td>
<td>$O(n^{4\frac{2}{3}})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theorem

If there is $\epsilon > 0$ such that 4-opt Detection admits an algorithm in time $O(n^{3-\epsilon} \cdot \text{polylog}(M))$, then there is $\delta > 0$ such that All Pairs Shortest Paths admits an algorithm in time $O(n^{3-\delta} \cdot \text{polylog}(M))$, assuming integer weights from $\{-M, \ldots, M\}$.
An equivalent representation of \( k \)-move

(The most intuitive) representation of \( k \)-move

A pair \((E^-, E^+)\), where \( E^- \subseteq H, E^+ \subseteq E(G) \)

\[
\begin{align*}
E^- &= \{1, 2, 3, 4, 5\} \\
E^+ &= \{6, 7, 8, 9, 10\}
\end{align*}
\]

\[
M = \{13, 25, 46\}
\]

A more useful representation: a pair \((f, M)\)

\(\Rightarrow\) an embedding \(f: \{1, \ldots, k\} \to \{1, \ldots, n\}\)

\(\Rightarrow\) connection pattern: a perfect matching \(M\) on \(\{1, \ldots, 2k\}\)

Łukasz Kowalik

From theory to practice in \( k \)-OPT heuristic for TSP
An equivalent representation of $k$-move

(The most intuitive) representation of $k$-move

A pair $(E^-, E^+)$, where $E^- \subseteq H, E^+ \subseteq E(G)$
An equivalent representation of $k$-move

(The most intuitive) representation of $k$-move

A pair $(E^-, E^+)$, where $E^- \subseteq H$, $E^+ \subseteq E(G)$

A more useful representation: a pair $(f, \ M)$

- an embedding $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$

```
f(1) = 2
f(2) = 5
f(3) = 9
```
An equivalent representation of $k$-move

(The most intuitive) representation of $k$-move

A pair $(E^-, E^+)$, where $E^- \subseteq H, E^+ \subseteq E(G)$

\[
\begin{align*}
&f(1) = 2 \\
&f(2) = 5 \\
&f(3) = 9 \\
&M = \{13, 25, 46\}
\end{align*}
\]

A more useful representation: a pair $(f, M)$

- an embedding $f : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$
- connection pattern: a perfect matching $M$ on $\{1, \ldots, 2k\}$
de Berg et al.’s idea

Observation 1
Now we can specify a connection pattern $M$ before specifying an embedding $f$.

Observation 2
There are only $O((2k)!)$ connection patterns, i.e., $O(1)$ for fixed $k$. 

$f(1) = 2$
$f(2) = 5$
$f(3) = 9$

$M = \{13, 25, 46\}$
For each of the $O((2k)!)$ connection patterns $M$, find the embedding $f_M$ which maximizes weight improvement.

Fixing $M$ allows for exploiting the structure of the solution.

From now on, assume $M$ is fixed.
Key notion: the dependence graph $D_M$

$$V(D_M) = [k].$$

Vertex $i$ corresponds to the $i$-th deleted edge from the Hamiltonian cycle $e_1e_2\cdots e_n$.

$$E(D_M) = O \cup I_M,$$

where

$$O = \{12, 23, \ldots, (k-1)k\}$$

- Edge $j(j + 1) \in O$ represents the property $f(j) < f(j + 1)$.
- $I_M$ is defined by $M$. Edge $ij \in I_M$ means that the cost of embedding $i$-the edge depends on $f(j)$. 
\( E(D_M) = O \cup I_M \)

- \( O = \{12, 23, \ldots, (k - 1)k\} \)

- Get \( I_M \) from \( M \) by identifying \( 2i - 1 \) with \( 2i \) for \( i \in [k] \):
  \[ I_M = \{ij : i'j' \in M, i' \in \{2i - 1, 2i\}, j' \in \{2j - 1, 2j\}\} \]

\[ D = ([3], O \cup I_M) \]
\[ E(D_M) = O \cup I_M \]

- \( O = \{12, 23, \ldots, (k - 1)k\} \)

- Get \( I_M \) from \( M \) by identifying \( 2i - 1 \) with \( 2i \) for \( i \in [k] \):
  \[ I_M = \{ij : i'j' \in M, i' \in \{2i - 1, 2i\}, j' \in \{2j - 1, 2j\}\} \]
The algorithm of de Berg, Buchin, Jansen and Woeginger

1. Find a minimum vertex cover $A$ of $I_M$
2. Embed $A$ in all $n_{|A|}$ ways
3. Dependence graph of the rest $D'$ has only some edges of $O$. $D'$ is a collection of paths so we can find optimal embedding in $O(nk)$ time using dynamic programming.

We have $|A| \leq \lfloor \frac{2}{3}k \rfloor$ (worst case: $I_M$ is a collection of 3-cycles).
Hence, time is $O(n\lfloor \frac{2}{3}k \rfloor + 1)k$ for every connection pattern.

$Lukasz Kowalik$

From theory to practice in k-OPT heuristic for TSP
The algorithm of de Berg, Buchin, Jansen and Woeginger

1. Find a minimum vertex cover $A$ of $I_M$

2. Embed $A$ in all $n! |A|$ ways

3. Dependence graph of the rest $D'$ has only some edges of $O$. $D'$ is a collection of paths so we can find optimal embedding in $O(nk)$ time using dynamic programming.

We have $|A| \leq \left\lfloor \frac{2}{3} k \right\rfloor$ (worst case: $I_M$ is a collection of 3-cycles).

Hence, time is $O(n \left\lfloor \frac{2}{3} k \right\rfloor + 1) k$ for every connection pattern.
The algorithm of de Berg, Buchin, Jansen and Woeginger

\[ D = ([4], O \cup I_M) \]

1. Find a minimum vertex cover \( A \) of \( I_M \)
2. Embed \( A \) in all \( n^{|A|} \) ways
The algorithm of de Berg, Buchin, Jansen and Woeginger

1. Find a minimum vertex cover $A$ of $I_M$
2. Embed $A$ in all $n^{|A|}$ ways
3. Dependence graph of the rest $D'$ has only some edges of $O$. $D'$ is a collection of paths so we can find optimal embedding in $O(nk)$ time using dynamic programming.

We have $|A| \leq \lfloor 2/3k \rfloor$ (worst case: $I_M$ is a collection of 3-cycles).
Hence, time is $O(n^{\lfloor 2/3k \rfloor + 1}k)$ for every connection pattern.
Another possible algorithm

\[ D = ([4], O \cup I_M) \]
Another possible algorithm

\[ D = ([4], O \cup I_M) \]

1. Embed 2, 4, \ldots, 2^\lfloor k/2 \rfloor in all \( n^\lfloor k/2 \rfloor \) ways
Another possible algorithm

\[ D = ([4], O \cup I_M) \]

1. Embed 2, 4, \ldots, 2\lfloor k/2 \rfloor in all \( n^{\lfloor k/2 \rfloor} \) ways

2. Dependence graph of the rest \( D' \) has only some edges of \( I_M \). \( D' \) is a collection of \textbf{cycles and paths} so we can find optimal embedding in \( O(n^3) \) time using dynamic programming.

Hence, time is \( O(n^{\lfloor k/2 \rfloor + 3}) \) for every connection pattern.
DO NOT OVERSIMPLIFY THE DEPENDENCE GRAPH!

EXPLOIT ITS STRUCTURE!
Tree decompositions and treewidth

Graph $G = (V, E)$

Tree decomposition of $G$

Tree decomposition is a tree of bags (subsets of $V$)
Tree decompositions and treewidth

Graph $G = (V, E)$

Tree decomposition of $G$

Tree decomposition is a tree of **bags** (subsets of $V$) such that

- For every edge $uv \in E$ some bag contains $u$ and $v$
Tree decompositions and treewidth

Graph $G = (V, E)$

Tree decomposition of $G$

Tree decomposition is a tree of **bags** (subsets of $V$) such that

- For every edge $uv \in E$ some bag contains $u$ and $v$
- For every vertex $v \in V$ bags containing $v$ form nonempty subtree (connected!)
Tree decompositions and treewidth

Graph $G = (V, E)$

Tree decomposition of $G$

Tree decomposition is a tree of bags (subsets of $V$) such that

- For every edge $uv \in E$ some bag contains $u$ and $v$
- For every vertex $v \in V$ bags containing $v$ form nonempty subtree (connected!)

Width of the decomposition: maximum bag size $-1$ (here: 3).
Treewidth of $G$: minimum width of a decomposition of $G$. 

Łukasz Kowalik
From theory to practice in k-OPT heuristic for TSP
Dynamic programming

For every node $t$ of a tree decomposition of the graph $D_M$:

- $X_t =$ the bag at $t$,
- $V_t =$ union of all bags in the subtree rooted at $t$.

For every node $t$ and partial embedding $f : X_t \rightarrow [n]$, compute

$$T_t[f] = \max_{g : V_t \rightarrow [n]} \text{gain}_M(g).$$

in the bottom-up fashion.
Dynamic programming: example

\[ D = ([4], O \cup I_M) \]

\[
T_{123}[f] = w(e_f(1)) + w(e_f(2)) + w(e_f(3)) - w(E^{+}_{f,M})
\]

\[
T_{23}[f] = \max_{g: \{1,2,3\} \rightarrow [n]} T_{123}[g].
\]

\[
T_{234}[f] = T_{23}[f|_{\{2,3\}}] + w(e_f(4)) - w(E^{+}_{f,M} \setminus E^{+}_{f|_{\{2,3\}},M})
\]
The $O(n^{(1/3+\epsilon_k)k})$-time algorithm

Theorem

Given a connection pattern $M$, the best $k$-move $(f, M)$ can be found in time $n^{\text{tw}(D_M) + 1}k^2 + 2^k$.

Theorem (Fomin et al. 2009)

Treewidth a $k$-vertex graph of maximum degree 4 is bounded by $(\frac{1}{3} + \epsilon_k)k$, where $\lim_{k \to \infty} \epsilon_k = 0$.

Corollary

For every fixed integer $k$, $k$-OPT Optimization can be solved in time $O(n^{(1/3+\epsilon_k)k})$, where $\lim_{k \to \infty} \epsilon_k = 0$.
One more idea: bucketing

Divide the $n$ edges of the Hamiltonian cycle into $n^{1/4}$ buckets of size $s = n^{3/4}$.
One more idea: bucketing

Divide the $n$ edges of the Hamiltonian cycle into $n^{1/4}$ buckets of size $s = n^{3/4}$.

Go through all assignments $b : [k] \rightarrow [n^{1/4}]$ of the $k$ edges to buckets.
One more idea: bucketing

Divide the \( n \) edges of the Hamiltonian cycle into \( n^{1/4} \) buckets of size \( s = n^{3/4} \).

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\circ & \circ & \circ & \circ \\
\end{array}
\]

\[
e_1, e_2, \ldots, e_s \hspace{1cm} e_{s+1}, \ldots, e_{2s} \hspace{1cm} e_{2s+1}, \ldots, e_{3s} \hspace{1cm} e_{3s+1}, \ldots, e_n
\]

Go through all assignments \( b: [k] \to [n^{1/4}] \) of the \( k \) edges to buckets.

- Edges of \( O \) in \( D_M \) between buckets no longer needed:
  
  \[
  D_{M,b} = ([4], O \cup I_M)
  \]
One more idea: bucketing

Divide the $n$ edges of the Hamiltonian cycle into $n^{1/4}$ buckets of size $s = n^{3/4}$.

Go through all assignments $b : [k] \rightarrow [n^{1/4}]$ of the $k$ edges to buckets.

- Edges of $O$ in $D_M$ between buckets no longer needed:

  \[
  D_{M,b} = ([4], O \cup I_M)
  \]

- Dynamic programming works faster, in time $O(n^{3/4}(\text{tw}(D_{M,b})+1))$
One more idea: bucketing

Divide the $n$ edges of the Hamiltonian cycle into $n^{1/4}$ buckets of size $s = n^{3/4}$.

\[
e_1, e_2, \ldots, e_s \quad e_{s+1}, e_{s+2}, \ldots, e_{2s} \quad e_{2s+1}, \ldots, e_{3s} \quad e_{3s+1}, \ldots, e_n
\]

Go through all assignments $b : [k] \rightarrow [n^{1/4}]$ of the $k$ edges to buckets.

- Edges of $O$ in $D_M$ between buckets no longer needed:

\[
D_{M,b} = ([4], O \cup I_M)
\]

- Dynamic programming works faster, in time $O(n^{3 \cdot (\text{tw}(D_{M,b})+1)})$

- Price: many bucket assignments to consider.
The $O(n^{(1/4+\epsilon_k)k})$-time algorithm

Plugging in the bucketing idea gives our main result (calculations skipped).

**Theorem**

*For every fixed integer $k$, $k$-OPT OPTIMIZATION can be solved in time $O(n^{(1/4+\epsilon_k)k})$, where $\lim_{k \to \infty} \epsilon_k = 0$.***
Part II: Practice
(joint work with Marek Cygan and Kamil Żyła)
A new heuristic based on theory

\((k, t)\)-OPT heuristic

For every connection pattern \( M \), find the best \( k \)-move, restricted only to the \( k \)-moves with dependence graph \( D_M \) of treewidth at most \( t \).
Some statistics

<table>
<thead>
<tr>
<th>tw</th>
<th>$k = 2$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>11</td>
<td>37</td>
<td>106</td>
<td>334</td>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>11</td>
<td>90</td>
<td>645</td>
<td>4423</td>
<td>29234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>71</td>
<td>1444</td>
<td>22303</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some thoughts

Possible goals
- (minor:) show an improvement over 2-OPT, 3-OPT,
- (major:) add as an element of a state-of-the-art solver, check if it helps.

Some ideas
- A question to address: How one should control $k$, $t$ (bound on treewidth) and $n$ (length of a fragment of the tour) during the whole local search process? (Increase parameters when stuck?)
- Make sure space usage is relatively low (embed some edges by brute-force? use bucketing?)