On the Max k-Vertex Cover problem

F. Della Croce, V. Th. Paschos

federico.dellacroce@polito.it
paschos@lamsade.dauphine.fr

Outline

Introduction

- **Max k-Vertex problem** (also known as partial vertex cover): given a graph $G(V, E)$ of order n and a constant $k < n$, we search for k vertices that cover the maximum number of edges in G.
- **Complexity**:
  NP-hard (straightforward reduction from the minimum vertex cover problem).
- **Notation**:
  We use notation $O^*(\cdot)$ to measure the running time of an algorithm ignoring polynomial factors.

Introduction

- **Max k-Vertex problem** (also known as partial vertex cover): given a graph $G(V, E)$ of order n and a constant $k < n$, we search for k vertices that cover the maximum number of edges in G.
- **Complexity**:
  NP-hard (straightforward reduction from the minimum vertex cover problem).
- **Notation**:
  We use notation $O^*(\cdot)$ to measure the running time of an algorithm ignoring polynomial factors.

Introduction

- **Max k-Vertex problem** (also known as partial vertex cover): given a graph $G(V, E)$ of order n and a constant $k < n$, we search for k vertices that cover the maximum number of edges in G.
- **Complexity**:
  NP-hard (straightforward reduction from the minimum vertex cover problem).
- **Notation**:
  We use notation $O^*(\cdot)$ to measure the running time of an algorithm ignoring polynomial factors.

Literature

- Apparently no polynomial-space exact approaches availables with complexity lower than the trivial 2^n. An exact algorithm with complexity $O^*(n^{0.792k})$ was proposed by Cai (Computer Journal 08).
- **Fixed parameter tractability**:
  - **W[1]**-hard wrt parameter k (very little hope that an algorithm solving Max k-Vertex problem with running time $O^*(f(k))$ could ever be devised) - Guo et al., WADS05.
  - in FPT wrt standard parameter l (1.396^l - Kneis et al., WG08).
  - in FPT wrt parameter w (w = graph treewidth: 2^w - Moser, 05).

Literature

- Apparently no polynomial-space exact approaches availables with complexity lower than the trivial 2^n. An exact algorithm with complexity $O^*(n^{0.792k})$ was proposed by Cai (Computer Journal 08).
- **Fixed parameter tractability**:
  - **W[1]**-hard wrt parameter k (very little hope that an algorithm solving Max k-Vertex problem with running time $O^*(f(k))$ could ever be devised) - Guo et al., WADS05.
  - in FPT wrt standard parameter l (1.396^l - Kneis et al., WG08).
  - in FPT wrt parameter w (w = graph treewidth: 2^w - Moser, 05).

Literature

- Apparently no polynomial-space exact approaches availables with complexity lower than the trivial 2^n. An exact algorithm with complexity $O^*(n^{0.792k})$ was proposed by Cai (Computer Journal 08).
- **Fixed parameter tractability**:
  - **W[1]**-hard wrt parameter k (very little hope that an algorithm solving Max k-Vertex problem with running time $O^*(f(k))$ could ever be devised) - Guo et al., WADS05.
  - in FPT wrt standard parameter l (1.396^l - Kneis et al., WG08).
  - in FPT wrt parameter w (w = graph treewidth: 2^w - Moser, 05).

Literature

- Apparently no polynomial-space exact approaches availables with complexity lower than the trivial 2^n. An exact algorithm with complexity $O^*(n^{0.792k})$ was proposed by Cai (Computer Journal 08).
- **Fixed parameter tractability**:
  - **W[1]**-hard wrt parameter k (very little hope that an algorithm solving Max k-Vertex problem with running time $O^*(f(k))$ could ever be devised) - Guo et al., WADS05.
  - in FPT wrt standard parameter l (1.396^l - Kneis et al., WG08).
  - in FPT wrt parameter w (w = graph treewidth: 2^w - Moser, 05).

An exact approach combining Branch-and-reduce and Measure-and-conquer (I)

- **Branch-and-reduce:**
  Search tree algorithm where at each branch the problem size is typically reduced in terms of variables.
- **Measure-and-conquer:** (Fomin et al. JACM09):
  weights are typically added to the variables and such weights opportunately decrease when the search tree levels increase.
- **Branching scheme**: vertex j is either selected or discarded.
- **Remark**: with a pure branch-and-reduce approach, no improvement occurs wrt the trivial $O^*(2^n)$.

An exact approach combining Branch-and-reduce and Measure-and-conquer (I)

- **Branch-and-reduce:**
  Search tree algorithm where at each branch the problem size is typically reduced in terms of variables.
- **Measure-and-conquer:** (Fomin et al. JACM09):
  weights are typically added to the variables and such weights opportunately decrease when the search tree levels increase.
- **Branching scheme**: vertex j is either selected or discarded.
- **Remark**: with a pure branch-and-reduce approach, no improvement occurs wrt the trivial $O^*(2^n)$.

An exact approach combining Branch-and-reduce and Measure-and-conquer (I)

- **Branch-and-reduce:**
  Search tree algorithm where at each branch the problem size is typically reduced in terms of variables.
- **Measure-and-conquer:** (Fomin et al. JACM09):
  weights are typically added to the variables and such weights opportunately decrease when the search tree levels increase.
- **Branching scheme**: vertex j is either selected or discarded.
- **Remark**: with a pure branch-and-reduce approach, no improvement occurs wrt the trivial $O^*(2^n)$.

An exact approach combining Branch-and-reduce and Measure-and-conquer (I)

- **Branch-and-reduce:**
  Search tree algorithm where at each branch the problem size is typically reduced in terms of variables.
- **Measure-and-conquer:** (Fomin et al. JACM09):
  weights are typically added to the variables and such weights opportunately decrease when the search tree levels increase.
- **Branching scheme**: vertex j is either selected or discarded.
- **Remark**: with a pure branch-and-reduce approach, no improvement occurs wrt the trivial $O^*(2^n)$.

An exact approach combining Branch-and-reduce and Measure-and-conquer (II)

- **Further notation**:
  - d_j = degree of vertex j
  - $N(j)$ the subset of vertices adjacent to j.
  - α_j = total number of vertices adjacent to j that have been discarded in the previous levels of the search tree.
  - $c_j = d_j - \alpha_j$.
- **Remark**: - Branch on j -
  for each $l \in N(j)$,
  if j is selected, then d_l is decreased by one unit as edge (j, l) is already covered by j;
  if j is discarded, then d_l is not modified and α_l is increased by one unit.

An exact approach combining Branch-and-reduce and Measure-and-conquer (II)

- **Further notation**:
  - d_j = degree of vertex j
  - $N(j)$ the subset of vertices adjacent to j.
  - α_j = total number of vertices adjacent to j that have been discarded in the previous levels of the search tree.
  - $c_j = d_j - \alpha_j$.
- **Remark**: - Branch on j -
  for each $l \in N(j)$,
  if j is selected, then d_l is decreased by one unit as edge (j, l) is already covered by j;
  if j is discarded, then d_l is not modified and α_l is increased by one unit.

An exact approach combining Branch-and-reduce and Measure-and-conquer (II)

- **Further notation**:
  - d_j = degree of vertex j
  - $N(j)$ the subset of vertices adjacent to j.
  - α_j = total number of vertices adjacent to j that have been discarded in the previous levels of the search tree.
  - $c_j = d_j - \alpha_j$.
- **Remark**: - Branch on j -
  for each $l \in N(j)$,
  if j is selected, then d_l is decreased by one unit as edge (j, l) is already covered by j;
  if j is discarded, then d_l is not modified and α_l is increased by one unit.

An exact approach combining Branch-and-reduce and Measure-and-conquer (III)

- **Measure and conquer**:
  we do not count in the measure the fixed vertices (those that have been either selected or discarded at an earlier stage of the search tree) and we count with a weight w_h the free (unfixed) vertices h.

An exact approach combining Branch-and-reduce and Measure-and-conquer (IV)

- **Weights of the free vertices**:
  each free vertex h is assigned a weight $w_h = w_{[i]}$ with $i = c_i = d_h - \alpha_h$ and we impose $w_{[0]} \leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[c_{\max}]} = 1$ that is the weights of the vertices are strictly increasing in their c_j coefficients.
- **Branching decision**
  We branch on the vertex j with largest coefficient $c_j = d_j - \alpha_j$ (notice that $c_{\max} \leq \Delta$).

An exact approach combining Branch-and-reduce and Measure-and-conquer (IV)

- **Weights of the free vertices**:
  each free vertex h is assigned a weight $w_h = w_{[i]}$ with $i = c_i = d_h - \alpha_h$ and we impose $w_{[0]} \leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[c_{\max}]} = 1$ that is the weights of the vertices are strictly increasing in their c_j coefficients.
- **Branching decision**
  We branch on the vertex j with largest coefficient $c_j = d_j - \alpha_j$ (notice that $c_{\max} \leq \Delta$).

An exact approach combining Branch-and-reduce and Measure-and-conquer (V)

- We get recurrences on the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of its vertices.
- For $p = 0$, there are only vertices with weight w_0 in the graph and, in this case, the problem is immediately solved by selecting the $k - \gamma$ vertices with largest α_j (if $\gamma < k$ vertices have been selected so far). Hence, free vertices j with no adjacent free vertices receive weight $w_[0] = 0$.

An exact approach combining Branch-and-reduce and Measure-and-conquer (V)

- We get recurrences on the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of its vertices.
- For $p = 0$, there are only vertices with weight w_0 in the graph and, in this case, the problem is immediately solved by selecting the $k - \gamma$ vertices with largest α_j (if $\gamma < k$ vertices have been selected so far). Hence, free vertices j with no adjacent free vertices receive weight $w_[0] = 0$.

The exact algorithm MAXKVC (I)

- MAXKVC:
  Select j such that c_j is maximum and branch according to the following exhaustive cases:
  1. if $c_j \geq 3$, then branch on j and either select or discard j;
  2. else, $c_j \leq 2$ and MAXKVC is polynomially solvable (by dynamic programming - Niedermeier et al. JA03).

Theorem

Algorithm MAXKVC solves the Max k-Vertex Cover problem with running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

The exact algorithm MAXKVC (I)

- MAXKVC:
  Select j such that c_j is maximum and branch according to the following exhaustive cases:
  1. if $c_j \geq 3$, then branch on j and either select or discard j;
  2. else, $c_j \leq 2$ and MAXKVC is polynomially solvable (by dynamic programming - Niedermeier et al. JA03).

Theorem

Algorithm MAXKVC solves the Max k-Vertex Cover problem with running time $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

The exact algorithm MAXKVC (II)

Sketch of proof

- Branch on the vertex j with largest $c_j = c_{\max} \leq \Delta$ where $c_j \geq 3$ and either we select or discard j.
- If we select j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease degree and coefficient by one unit.
- If we discard j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their coefficient as their α parameter is increased by one unit.

The exact algorithm MAXKVC (II)

Sketch of proof

- Branch on the vertex j with largest $c_j = c_{\max} \leq \Delta$ where $c_j \geq 3$ and either we select or discard j.
- If we select j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease degree and coefficient by one unit.
- If we discard j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their coefficient as their α parameter is increased by one unit.

The exact algorithm MAXKVC (II)

Sketch of proof

- Branch on the vertex j with largest $c_j = c_{\max} \leq \Delta$ where $c_j \geq 3$ and either we select or discard j.
- If we select j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease degree and coefficient by one unit.
- If we discard j, vertex j is fixed and $c_{\max}$ vertices (the neighbors of j) decrease their coefficient as their α parameter is increased by one unit.

The exact algorithm MAXKVC (III)

- The recurrence is
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - \sum_{h \in N(j)} \left(w_{[c_h]} - w_{[c_h - 1]}\right)\right)$
- By constraining the weights to satisfy the inequality $w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}$, $\forall j = 2, \ldots, c_{\max}$, the recurrence becomes in the worst-case
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - c_{\max}\left(w_{[c_{\max}]} - w_{[c_{\max}-1]}\right)\right)$.
- As $c_{\max} \leq \Delta$, where the equality occurs when $\alpha_j = 0$, the above recurrence becomes, in the worst-case,
  $T(p) \leq 2T\left(p - w_{[\Delta]} - \Delta\left(w_{[\Delta]} - w_{[\Delta-1]}\right)\right)$.

The exact algorithm MAXKVC (III)

- The recurrence is
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - \sum_{h \in N(j)} \left(w_{[c_h]} - w_{[c_h - 1]}\right)\right)$
- By constraining the weights to satisfy the inequality $w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}$, $\forall j = 2, \ldots, c_{\max}$, the recurrence becomes in the worst-case
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - c_{\max}\left(w_{[c_{\max}]} - w_{[c_{\max}-1]}\right)\right)$.
- As $c_{\max} \leq \Delta$, where the equality occurs when $\alpha_j = 0$, the above recurrence becomes, in the worst-case,
  $T(p) \leq 2T\left(p - w_{[\Delta]} - \Delta\left(w_{[\Delta]} - w_{[\Delta-1]}\right)\right)$.

The exact algorithm MAXKVC (III)

- The recurrence is
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - \sum_{h \in N(j)} \left(w_{[c_h]} - w_{[c_h - 1]}\right)\right)$
- By constraining the weights to satisfy the inequality $w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}$, $\forall j = 2, \ldots, c_{\max}$, the recurrence becomes in the worst-case
  $T(p) \leq 2T\left(p - w_{[c_{\max}]} - c_{\max}\left(w_{[c_{\max}]} - w_{[c_{\max}-1]}\right)\right)$.
- As $c_{\max} \leq \Delta$, where the equality occurs when $\alpha_j = 0$, the above recurrence becomes, in the worst-case,
  $T(p) \leq 2T\left(p - w_{[\Delta]} - \Delta\left(w_{[\Delta]} - w_{[\Delta-1]}\right)\right)$.

The exact algorithm MAXKVC (IV)

All together we want to satisfy contemporaneously

- **the recurrences**
  $T(p) \leq 2T(p - w_{[i]} - i(w_{[i]} - w_{[i-1]}))$, $\forall i \in 3, \ldots, \Delta$ (as $c_j \geq 3$)
- **the constraints**

$$\begin{aligned} w_{[i]} - w_{[i-1]} &\leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ 0 = w_{[0]} &\leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

This corresponds to a non linear optimization problem of the form:

$$\begin{aligned} &\min \ \alpha \\ &\alpha^{(w_{[i]} + i(w_{[i]} - w_{[i-1]}))} \geq 2 \quad \forall i = 3, \ldots, \Delta \\ &w_{[i]} - w_{[i-1]} \leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ &0 = w_{[0]} \leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

The exact algorithm MAXKVC (IV)

All together we want to satisfy contemporaneously

- **the recurrences**
  $T(p) \leq 2T(p - w_{[i]} - i(w_{[i]} - w_{[i-1]}))$, $\forall i \in 3, \ldots, \Delta$ (as $c_j \geq 3$)
- **the constraints**

$$\begin{aligned} w_{[i]} - w_{[i-1]} &\leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ 0 = w_{[0]} &\leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

This corresponds to a non linear optimization problem of the form:

$$\begin{aligned} &\min \ \alpha \\ &\alpha^{(w_{[i]} + i(w_{[i]} - w_{[i-1]}))} \geq 2 \quad \forall i = 3, \ldots, \Delta \\ &w_{[i]} - w_{[i-1]} \leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ &0 = w_{[0]} \leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

The exact algorithm MAXKVC (IV)

All together we want to satisfy contemporaneously

- **the recurrences**
  $T(p) \leq 2T(p - w_{[i]} - i(w_{[i]} - w_{[i-1]}))$, $\forall i \in 3, \ldots, \Delta$ (as $c_j \geq 3$)
- **the constraints**

$$\begin{aligned} w_{[i]} - w_{[i-1]} &\leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ 0 = w_{[0]} &\leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

This corresponds to a non linear optimization problem of the form:

$$\begin{aligned} &\min \ \alpha \\ &\alpha^{(w_{[i]} + i(w_{[i]} - w_{[i-1]}))} \geq 2 \quad \forall i = 3, \ldots, \Delta \\ &w_{[i]} - w_{[i-1]} \leq w_{[i-1]} - w_{[i-2]} \quad \forall i = 2, \ldots, \Delta \\ &0 = w_{[0]} \leq w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[\Delta-1]} \leq w_{[\Delta]} = 1 \end{aligned}$$

The exact algorithm MAXKVC (V)

By means of a non linear solver, we get the performances of the algorithm for small values of Δ.

Δ	3	4	5	6	7	8	9
$O^*(n)$	1.414^n	1.516^n	1.587^n	1.641^n	1.682^n	1.714^n	1.741^n

For all values of Δ, the complexity corresponds to $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.
This is not accidental:, by setting:

$$w_{[i]} = \frac{(i-1)(\Delta+1)}{(i+1)(\Delta-1)} \quad \forall i = 2, \ldots, \Delta$$

$$w_{[1]} = \frac{1}{2} w_{[2]}, \quad w_{[0]} = 0$$

the constraints are satisfied and the complexity is $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$

The exact algorithm MAXKVC (V)

By means of a non linear solver, we get the performances of the algorithm for small values of Δ.

Δ	3	4	5	6	7	8	9
$O^*(n)$	1.414^n	1.516^n	1.587^n	1.641^n	1.682^n	1.714^n	1.741^n

For all values of Δ, the complexity corresponds to $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

This is not accidental:, by setting:

$$w_{[i]} = \frac{(i-1)(\Delta+1)}{(i+1)(\Delta-1)} \quad \forall i = 2, \ldots, \Delta$$

$$w_{[1]} = \frac{1}{2} w_{[2]}, \quad w_{[0]} = 0$$

the constraints are satisfied and the complexity is $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$

The exact algorithm MAXKVC (V)

By means of a non linear solver, we get the performances of the algorithm for small values of Δ.

Δ	3	4	5	6	7	8	9
$O^*(n)$	1.414^n	1.516^n	1.587^n	1.641^n	1.682^n	1.714^n	1.741^n

For all values of Δ, the complexity corresponds to $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.
This is not accidental:, by setting:

$$\begin{aligned} w_{[i]} &= \frac{(i-1)(\Delta+1)}{(i+1)(\Delta-1)} \quad \forall i = 2, \ldots, \Delta \\ w_{[1]} &= \frac{1}{2} w_{[2]}, \quad w_{[0]} = 0 \end{aligned}$$

the constraints are satisfied and the complexity is $O^*(2^{\frac{\Delta-1}{\Delta+1}n})$.

Tailoring for $\Delta = 3$

Proposition

Algorithm `MAXKVC-3` *solves the Max k-Vertex Cover Problem on graphs with maximum degree 3 with running time* $O^*(1.3339^n)$.

Parameterization by τ

Theorem

Max k-Vertex Cover Problem is solvable with running time $O^*(2^\tau)$ where τ is the size of the minimum vertex cover.

1. compute a minimum vertex cover V' $O^*(1.2738^\tau)$;
2. for every subset $S' \subseteq V'$ of cardinality at most k, take the $k - |S'|$ vertices of $V \setminus V'$ with the largest degrees to $V' \setminus S'$; denote by I' this latter set $(O^*(\sum_{i=1}^{k} \binom{\tau}{i}) < O^*(2^\tau))$;
3. return the best among the sets $S' \cup I'$ so-computed (i.e., the set that covers the maximum of edges).

Parameterization by τ

Theorem

Max k-Vertex Cover Problem is solvable with running time $O^*(2^\tau)$ where τ is the size of the minimum vertex cover.

1. compute a minimum vertex cover V' $O^*(1.2738^\tau)$;
2. for every subset $S' \subseteq V'$ of cardinality at most k, take the $k - |S'|$ vertices of $V \setminus V'$ with the largest degrees to $V' \setminus S'$; denote by I' this latter set $(O^*(\sum_{i=1}^{k} \binom{\tau}{i}) < O^*(2^\tau))$;
3. return the best among the sets $S' \cup I'$ so-computed (i.e., the set that covers the maximum of edges).

Parameterization by τ

Theorem

Max k-Vertex Cover Problem is solvable with running time $O^*(2^\tau)$ where τ is the size of the minimum vertex cover.

1. compute a minimum vertex cover V' $O^*(1.2738^\tau)$;
2. for every subset $S' \subseteq V'$ of cardinality at most k, take the $k - |S'|$ vertices of $V \setminus V'$ with the largest degrees to $V' \setminus S'$; denote by I' this latter set $(O^*(\sum_{i=1}^{k} \binom{\tau}{i}) < O^*(2^\tau))$;
3. return the best among the sets $S' \cup I'$ so-computed (i.e., the set that covers the maximum of edges).

Parameterization by τ

Theorem

Max k-Vertex Cover Problem is solvable with running time $O^*(2^\tau)$ where τ is the size of the minimum vertex cover.

1. compute a minimum vertex cover V' $O^*(1.2738^\tau)$;
2. for every subset $S' \subseteq V'$ of cardinality at most k, take the $k - |S'|$ vertices of $V \setminus V'$ with the largest degrees to $V' \setminus S'$; denote by I' this latter set $(O^*(\sum_{i=1}^{k} \binom{\tau}{i}) < O^*(2^\tau))$;
3. return the best among the sets $S' \cup I'$ so-computed (i.e., the set that covers the maximum of edges).

Further results

- Max k-Vertex Cover can be solved in time $O^*(\max\{\gamma^\tau, c^k\})$, for two constants $\gamma < 2$ and $c > 4$, and with polynomial space.
- Max k-Vertex Cover can be solved to optimality in $O^*(\Delta^k)$ and polynomial space.
- Max k-Vertex Cover can be approximated within ratio $1 - \epsilon$ and with running time:

$$\min\left\{O^*\left(n^{\left(1+2\sqrt{1-3\epsilon}\right)(\omega k)/9}\right), O^*\left(\frac{\tau}{\left(1+2\sqrt{1-3\epsilon}\right)k/3}\right)\right\}$$

and polynomial space.

Further results

- Max k-Vertex Cover can be solved in time $O^*(\max\{\gamma^\tau, c^k\})$, for two constants $\gamma < 2$ and $c > 4$, and with polynomial space.
- Max k-Vertex Cover can be solved to optimality in $O^*(\Delta^k)$ and polynomial space.
- Max k-Vertex Cover can be approximated within ratio $1 - \epsilon$ and with running time:

$$\min\left\{O^*\left(n^{\left(1+2\sqrt{1-3\epsilon}\right)(\omega k)/9}\right), O^*\left(\tau^{\left(1+2\sqrt{1-3\epsilon}\right)k/3}\right)\right\}$$

and polynomial space.

Further results

- Max k-Vertex Cover can be solved in time $O^*(\max\{\gamma^\tau, c^k\})$, for two constants $\gamma < 2$ and $c > 4$, and with polynomial space.
- Max k-Vertex Cover can be solved to optimality in $O^*(\Delta^k)$ and polynomial space.
- Max k-Vertex Cover can be approximated within ratio $1 - \epsilon$ and with running time:

$$\min\left\{O^*\left(n^{\left(1+2\sqrt{1-3\epsilon}\right)(\omega k)/9}\right), O^*\left(\frac{\tau}{\left(1+2\sqrt{1-3\epsilon}\right)k/3}\right)\right\}$$

and polynomial space.

Final issue

Do you know anything about *Min* k-Vertex Cover?