Chapter 3
Complexity and Algorithmic Results

3.1 Introduction

In this chapter we discuss complexity and algorithmic results on total domination in graphs. This area, although well studied, is still not developed to the same level as for domination in graphs. We will outline a few of the best-known algorithms and state what is currently known in this field.

3.2 Complexity

The basic complexity question concerning the decision problem for the total domination number takes the following form:

Total Dominating Set
Instance: A graph $G = (V, E)$ and a positive integer k
Question: Does G have a TD-set of cardinality at most k?

For a definition of the graph classes mentioned in this chapter, we refer the reader to the excellent survey on graph classes by Brandstädt, Le, and Spinrad [18] which is regarded as a definitive encyclopedia for the literature on graph classes.

Let graph class $\mathscr{G}_1$ be a proper subclass of a graph class $\mathscr{G}_2$, i.e., $\mathscr{G}_1 \subset \mathscr{G}_2$. If problem $\mathscr{P}$ is NP-complete when restricted to $\mathscr{G}_1$, then it is NP-complete on $\mathscr{G}_2$. Furthermore, any polynomial time algorithm that solves a problem $\mathscr{P}$ on $\mathscr{G}_2$ also solves $\mathscr{P}$ on $\mathscr{G}_1$. Hence it is useful to know the containment relations between certain graph classes. In particular, we have the containment relations described above Table 3.1:

Table 3.1 summarizes the NP-completeness results for the total domination number with the corresponding citation. We abbreviate "NP-complete" by "**NP-c**" and "polynomial time solvable" by "**P**."

M.A. Henning and A. Yeo, *Total Domination in Graphs*, Springer Monographs in Mathematics, DOI 10.1007/978-1-4614-6525-6_3,

$$
\begin{aligned}
\text{Bipartite} &\subset \text{comparability} \\
\text{Split} &\subset \text{chordal} \\
\text{Claw-free} &\subset \text{line} \\
\text{Strongly chordal} &\subset \text{dually chordal} \\
\text{Permutation} &\subset k\text{-polygon} \subset \text{circle} \\
\text{Interval} &\subset \text{strongly chordal} \subset \text{chordal} \\
\text{Permutation} &\subset \text{cocomparability} \subset \text{asteroidal triple-free}
\end{aligned}
$$

Table 3.1 NP-complete results for the total domination number

Graph Class	NP-completeness result	Citation
General graph	**NP-c**	[170]
Bipartite graph	**NP-c**	[170]
Comparability graph	**NP-c**	[170]
Split graph	**NP-c**	[158]
Chordal graph	**NP-c**	[159]
Line graph	**NP-c**	[165]
line graph of bipartite graph	**NP-c**	[165]
Claw-free graph	**NP-c**	[165]
Circle graph	**NP-c**	[151]
Planar graph, max-degree 3	**NP-c**	[71]
Chordal bipartite graph	**P**	[155, 174]
Interval graph	**P**	[10, 11, 25, 150, 176]
Permutation graph	**P**	[17, 38, 155]
Strongly chordal graph	**P**	[24]
Dually chordal graph	**P**	[155]
Cocomparability graph	**P**	[154, 155]
Asteroidal triple-free graphs	**P**	[156]
DDP-graphs	**P**	[156]
Distance hereditary graph	**P**	[26, 155]
k-polygon graph (fixed $k \geq 3$)	**P**	[155]
Partial k-tree (fixed $k \geq 3$)	**P**	[8, 195]
Trapezoid graph	**P**	[156]

As far as we are aware the class of chordal bipartite graphs is the only class where the complexities of finding a minimum dominating set and a minimum TD-set vary. The problem of finding a minimum dominating set in chordal bipartite graphs is NP-hard, by a result in [41].

3.2.1 Time Complexities

In [155] Kratsch and Stewart give a transformation that implies that any algorithm for domination can be used for total domination for a wide variety of graph classes, such as permutation graphs, dually chordal graphs, and k-polygon graphs. In [155] it is pointed out that this gives an $O(nm^2)$ algorithm for computing a minimum cardinality TD-set in cocomparability graphs, improving on an $O(n^6)$ algorithm in [154]. It also implies an $O(n+m)$ and an $O(n\ln(\ln(n)))$ algorithm for permutation graphs.

In [156] Kratsch gives an $O(n^6)$ algorithm for total domination in asteroidal triple-free graphs. Asteroidal triple-free graphs form a large class of graphs containing interval, permutation, trapezoid, and cocomparability graphs, and therefore, there exist $O(n^6)$ algorithms for these graph classes as well. In fact, in [156] it is shown that there is a $O(n^7)$ algorithm for total domination for the larger class of DDP-graphs. A DDP-graph is a graph where every component has a dominating diametral path, which is a path whose length is equal to the diameter and whose vertex set dominates the graph. Any asteroidal triple-free graph is also a DDP-graph.

In [174] Pradhan gives an $O(n+m)$ algorithm for finding a minimum TD-set in chordal bipartite graphs. In [26] a linear algorithm is also given for distance hereditary graphs.

Bertossi and Gori [11] constructed an $O(n\ln n)$ algorithm for the total domination number of an interval graph of order n.

Adhar and Peng [1] presented efficient parallel algorithms for total domination in interval graphs, while Bertossi and Moretti [12] and Rao and Pandu Rangan [177] presented efficient parallel algorithms for total domination on circular-arc graphs.

In Sect. 3.5 we give a linear time algorithm for finding a minimum TD-set in trees.

3.3 Fixed Parameter Tractability

As determining the total domination number of a graph is NP-hard for most classes of graphs, we need to consider either non-polynomial algorithms or approximation algorithms or heuristics. Towards the end of the last millennium Downey and Fellows [58] introduced a new concept to handle NP-hardness, namely, fixed parameter tractability, which is often abbreviated to FPT.

A problem, $\mathscr{P}$, with size n and a parameter k is fixed parameter tractable (FPT) if it can be solved in time $O(f(k)n^c)$, for some function $f(k)$ not depending on n and some constant c not depending on n or k. For total domination the parameter used is normally the size of the solution. So the question is if there exists an algorithm with complexity $O(f(\gamma_t(G))n^c)$ to determine the total domination number of a graph G. Unfortunately, according to the theory of FPT, it is very unlikely that determining the total domination number is fixed parameter tractable for general graphs. In fact

Table 3.2 FPT-complexity results for the total domination number

Graph class	FPT complexity	Citation
General graphs	$W[2]$-hard	[58]
Graphs with girth 3 or 4*	$W[2]$-hard	[171] (Philip, G.[a])
Girth at least 5*	FPT	(Philip, G.[b])
Planar graph	FPT	[2,58]
Bounded treewidth*	FPT	[2]
d-degenerate graphs*	FPT	[5] (Alon, N.[c])
Bounded maximum degree*	FPT	

*We give some explanation below.
[a]personal communication.
[b]personal communication.
[c]personal communication.

it is shown that the problem is $W[2]$-hard, which according to FPT theory means that the problem is not FPT unless a very unlikely collapse of complexity classes occurs. We will not go into more depth in this area and refer the interested reader to [58] for more information about FPT.

In Table 3.2 we list for which classes of graphs the problem of determining the total domination number is FPT and for which classes it is unlikely to be FPT, by being $W[2]$-hard.

We next describe some of the graph classes listed in Table 3.2 and approaches to determine their FPT complexity.

Girth 3 or 4: In this case the same reductions that were used for connected domination in [171] can be used (Philip, G., personal communication).

Girth at Least Five: Let G be a graph with girth at least 5 and suppose we want to decide if $\gamma_t(G) \leq k$, for some k. Let $x \in V(G)$ be arbitrary and note that as the girth is at least five, the vertex x is the only vertex in G that totally dominates more than one vertex in $N_G(x)$. Therefore if $d_G(x) > k$ and $\gamma_t(G) \leq k$, then the vertex x must belong to every minimum TD-set in G. If $d_G(x) \leq k$, then every TD-set has to include at least one vertex from $N_G(x)$ (in order to totally dominate x), and so by trying all possibilities, we obtain a search tree with branching factor at most k and depth at most k, implying that we can find all TD-sets in G of size at most k in time $O(k^k(n+m))$. Therefore we can also decide if $\gamma_t(G) \leq k$ in the same time, implying that the problem is FPT.

Bounded Treewidth: The notion of treewidth was introduced by Robertson and Seymour [178] and plays an important role in their fundamental work on graph minors. We therefore give a brief description of this important notion. A tree decomposition of a graph $G = (V,E)$ is a pair $(\mathscr{Y},T)$ where T is a tree with $V(T) = \{1,2,\ldots,r\}$ (for some $r \geq 1$) and $\mathscr{Y} = \{Y_i \subseteq V(G) \mid i = 1,2,\ldots,r\}$ is a family of subsets of $V(G)$ such that the following holds: (1) $\cup_{i=1}^{r} Y_i = V(G)$; (2) for each edge $e = \{u,v\} \in E(G)$, there exists a Y_i, such that $\{u,v\} \subseteq Y_i$; and (3) for all $v \in V(G)$, the set of vertices $\{i \mid v \in Y_i\}$ forms a connected subtree of T.

Given a tree decomposition, which always exists as we could let $r = 1$ and $Y_1 = V(G)$, the width is $\max\{|Y_i|\colon i = 1,2,\ldots,r\} - 1$, and the treewidth of a graph is the smallest possible width that can be obtained by a tree decomposition. A graph is said to have bounded treewidth if this value is bounded by a constant. For example, if G is a tree its treewidth can be shown to be 1.

d-Degenerate Graphs: A graph is called d-degenerate if every induced subgraph has a vertex of degree at most d. It was shown in [5] that it is FPT to find a dominating set of size at most k (the parameter) in a d-degenerate graph. The approach in [5] can also be used to show that the equivalent problem for total domination is FPT (Alon, N., personal communication).

Bounded Maximum Degree: If the maximum degree of a graph is bounded by a constant d, then the graph is d-degenerate and therefore the problem is FPT by the equivalent result for d-degenerate graphs. Alternatively it is also easy to see that the problem is FPT by using a simple search tree. As the size of all neighborhoods is at most d, the branching factor of the search tree is at most d. Further the depth is at most k, giving us the desired FPT algorithm (as d is considered a constant).

3.4 Approximation Algorithms

Let $H_i = 1+1/2+1/3+\cdots+1/i$, which is the ith harmonic number. It is known that $\ln(i)+ec < H_i < \ln(i)+1/(2i)+ec$, where $ec = 0.57721\ldots$ is the Euler constant. Using a result on the problem *Minimum Set Cover* in [59], the following is shown in [34].

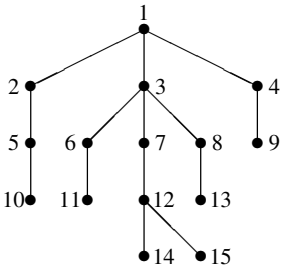
Theorem 3.1 ([34]). *There exists a $(H_{\Delta(G)} - \frac{1}{2})$-approximation algorithm for the problem TOTAL DOMINATION.*

Theorem 3.2 ([34]). *There exists constants $c > 0$ and $D \geq 3$ such that for every $\Delta \geq D$ it is NP-hard to approximate the problem TOTAL DOMINATION within a factor $\ln(\Delta) - c\ln\ln(\Delta)$ for bipartite graphs with maximum degree Δ.*

Note that by Theorem 3.2 it is NP-hard to approximate the problem TOTAL DOMINATION within a factor $\ln(\Delta) - c\ln\ln(\Delta)$ for general graphs with maximum degree $\Delta \geq 3$.

3.5 A Tree Algorithm

Laskar, Pfaff, Hedetniemi, and Hedetniemi [159] constructed the first linear algorithm for computing the total domination number of a tree. For ease of presentation, we consider rooted trees. We commonly draw the root of a rooted tree at the top with the remaining vertices at the appropriate level below the root depending



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
parent [0 1 1 1 2 3 3 3 4 5 6 7 8 12 12]

Fig. 3.1 A rooted tree T with its parent array

on their distance from the root. The linear algorithm we present here is similar to an algorithm due to Mitchell, Cockayne, and Hedetniemi [167] for computing the domination number of an arbitrary tree in the sense that we root the tree and systematically consider the vertices of the tree, starting from the vertices at furthest distance from the root, and carefully select a TD-set S in such a way that the sum of the distances from the root to the vertices in S is minimum. Given a rooted tree T with root vertex labeled 1 and with $V(T) = \{1,2,\ldots,n\}$, we represent T by a data structure called a *parent array* in which the parent of a vertex labeled i is given by parent$[i]$ with parent$[1] = 0$ (to indicate that the vertex labeled 1 has no parent). We assume that the vertices of T are labeled $1,2,\ldots,n$ so that for $i < j$, vertex i is at level less than or equal to that of vertex j (i.e., $d(1,i) \leq d(1,j)$). Figure 3.1 shows an example of a rooted tree T with its parent array.

In Algorithm TREE TOTAL DOMINATION that follows, we will for each vertex, i, keep track of two boolean values In_TD_Set$[i]$ and TD_by_child$[i]$, which both initially will be put to false. As the algorithm progresses some of these values may change to true. Upon completion of the algorithm, all vertices, i, with In_TD_Set$[i] =$ true will be added to the TD-set and all vertices with TD_by_child$[i] =$ true will be totally dominated by one of their children. Below we define vertex 0 to be a *dummy vertex* that does not exist and parent$[1] =$ parent$[0] = 0$. We will never change In_TD_Set$[0]$ but may change TD_by_child$[0]$.

Algorithm TREE TOTAL DOMINATION:

Input: A rooted tree $T = (V,E)$ with $V = \{1,2,\ldots,n\}$ rooted at 1 (where larger values are further from the root) and represented by an array parent$[1\ldots n]$
Output: An array In_TD_Set[] that indicates which vertices belong to the TD-set
Code:

1. **for** $i = 1$ **to** n **do** $\{$
2. TD_by_child$[i] =$ *false*
3. In_TD_Set$[i] =$ *false* $\}$
4. ***for*** $i = n$ ***to*** 2 ***do***
5. ***if*** (TD_by_child$[i] =$ *false*) ***and*** (In_TD_Set[parent$[i]] =$ *false*) ***then*** $\{$

6. In_TD_Set[parent[i]] = *true*
7. TD_by_child[parent[parent[i]]] = *true* }
8. ***if*** (TD_by_child[1] = *false*) ***then***
9. In_TD_Set[2] = *true*

We will now outline why the Algorithm TREE TOTAL DOMINATION does produce a minimum TD-set. Let $S(i)$ denote all vertices, j, with In_TD_Set[j] = true, just after performing line 7 with the value i. We will show that property $P(i)$ below holds for all $i = n, n-1, n-2, \ldots, 2$, by induction.

Property $P(i)$**:** $S(i)$ totally dominates $\{i, i+1, i+2, \ldots, n\}$ and $S(i)$ is a subset of some minimum TD-set in G. Furthermore the array TD_by_child is updated correctly.

Property $P(n)$ is clearly true as the set $S(n)$ only contains the unique neighbor of vertex n. So let $2 \leq i < n$ and assume that property $P(i+1)$ holds and that $S(i+1) \subseteq Q_{i+1}$, where Q_{i+1} is a minimum TD-set in G. We will now consider the two possible outcomes of line 5.

If TD_by_child[i] = false and In_TD_Set[parent[i]] = false, then some vertex, t, in Q_{i+1} must totally dominate i. If t = parent[i], then $S(i) \subseteq Q_{i+1}$, and therefore $S(i)$ is a subset of an optimal solution. Moreover as $S(i+1)$ totally dominates $\{i+1, i+2, \ldots, n\}$ and the vertex parent[i] $\in S(i)$ totally dominates the vertex i, the set $S(i)$ totally dominates $\{i, i+1, i+2, \ldots, n\}$. Hence we may assume that t is a child of i, for otherwise the desired result follows. Let $Q_i = (Q_{i+1} \setminus \{t\}) \cup \{\text{parent}[i]\}$ and note that Q_i is a minimum TD-set of G as all neighbors of t, except for the vertex i, are totally dominated by $S(i+1)$ and $S(i+1) \subseteq Q_i$. Further, $S(i) = S(i+1) \cup \{\text{parent}[i]\} \subseteq Q_i$, and so the set $S(i)$ is a subset of an optimal solution. As before since the set $S(i+1)$ totally dominates $\{i+1, i+2, \ldots, n\}$ and the vertex parent[i] $\in S(i)$ totally dominates the vertex i, the set $S(i)$ totally dominates $\{i, i+1, i+2, \ldots, n\}$.

If TD_by_child[i] = true or In_TD_Set[parent[i]] = true, then the vertex i is totally dominated by $S(i+1)$ and therefore $S(i)$ totally dominates $\{i, i+1, i+2, \ldots, n\}$. Furthermore $S(i) = S(i+1) \subseteq Q_{i+1}$ which implies that $S(i)$ is a subset of an optimal solution.

It remains for us to verify that the array TD_by_child is updated correctly. If $j \in S(i)$, then let i_j be the value of i when In_TD_Set[j] is put to true. That is, parent[i_j] = j. In line 7, TD_by_child[parent[j]] is then put to true. Furthermore suppose that TD_by_child[j'] = true for some vertex j'. Let i'_j be the value of i when TD_by_child[j'] was put to true. That is, parent[parent[i'_j]] = j'. In line 6, parent[i'_j] is then put to true, which implies that the child parent[i'_j] of the vertex j' belongs to the TD-set, as desired. This implies that TD_by_child is updated correctly. Therefore, Property $P(i)$ holds.

As Property $P(2)$ holds, we note that the set $S(2)$ is a subset of an optimal solution Q_2, say, and $S(2)$ totally dominates all vertices in G, except possibly for vertex 1. Therefore if vertex 1 is totally dominated by $S(2)$, then $S(2) = Q_2$ and S_2 is an optimal solution. On the other hand, if vertex 1 is not totally dominated by $S(2)$, then some vertex, z, in Q_2 must totally dominate vertex 1. Thus, z is a child of

vertex 1 and the set $S(1) = Q_1 = (Q_2 \setminus \{z\}) \cup \{2\}$ is an optimal solution. Therefore TREE TOTAL DOMINATION does indeed find an optimal solution.

It is also not difficult to see that the time complexity of the algorithm is $O(n)$ for trees of order n.

When we run our algorithm on the rooted tree T in Fig. 3.1, the following steps will be performed, yielding the set $\{12,8,7,6,5,4,3,2,1\}$ as a minimum TD-set in T, as the if statement in line 8 will be false.

i	Action
15	In_TD_Set[12] = true
14	None
13	In_TD_Set[8] = true
12	In_TD_Set[7] = true
11	In_TD_Set[6] = true

i	Action
10	In_TD_Set[5] = true
9	In_TD_Set[4] = true
8	In_TD_Set[3] = true
7	None
6	None

i	Action
5	In_TD_Set[2] = true
4	In_TD_Set[1] = true
3	None
2	None

3.6 A Simple Heuristic

In this section we will give a simple heuristic that finds a TD-set in a graph G of size at most $n(G)(1+\ln\delta(G))/\delta(G)$. Furthermore, in Theorem 3.4 it is shown that there exist graphs where there do not exist TD-sets of size much smaller than this.

The heuristic simply keeps choosing the vertex that totally dominates the maximum number of vertices which are not yet totally dominated and adding this vertex to our TD-set. It continues to do this until all vertices are totally dominated. Below is the code for this heuristic, where T will be our TD-set and Not_TD will contain the vertices that are not totally dominated by T.

Heuristic TOTAL DOMINATION:

Input: A graph $G = (V,E)$ with minimum degree $\delta \geq 1$ and order n.
Output: A TD-set T of G.
Code:

1. $T = \emptyset$
2. Not_TD $= V(G)$
3. ***While*** Not_TD $\neq \emptyset$ ***do*** $\{$
4. ***Let*** $x \in V(G)$ ***have maximum*** $d_{\text{Not\_TD}}(x)$
5. $T = T \cup \{x\}$
6. Not_TD $=$ Not_TD $\setminus N(x)$ $\}$
7. $\}$

The fact that Heuristic TOTAL DOMINATION finds a TD-set of size at most $n(G)(1+\ln\delta(G))/\delta(G)$ was shown in [130], by considering the open neighborhood hypergraph, H_G, of G. We will below give a proof of this result not involving hypergraphs.

It is not too difficult to show that with the correct implementation, the Heuristic TOTAL DOMINATION can be made to run in time $O(n+m)$. It would require us to keep a data structure (such as a hash table) that allows us to perform line 4 in constant time. We would then need to continuously update the degrees $d_{\text{Not\_TD}}()$ and note that we can decrease such degrees at most $2m(G)$ times (when removing vertices from Not_TD as $\sum_{v\in G} d_G(v) = 2m(G)$). We will leave the details of the complexity computations to the interested reader.

In fact if all we want is the bound $n(G)(1+\ln\delta(G))/\delta(G)$, then we can find a TD-set of size at most in this time $\mathrm{O}(n+\delta(G)n)$ by considering the open neighborhood hypergraph, H_G, of G and making it $\delta(G)$-uniform. However in practice the heuristic given in this section will generally perform better (when G is not regular).

Theorem 3.3. *If G is a graph with minimum degree $\delta \geq 1$, order n, and size m, then Heuristic TOTAL DOMINATION produces a TD-set T in G satisfying*

$$|T| \leq \left(\frac{1+\ln\delta}{\delta}\right) n.$$

The complexity of the algorithm is $\mathrm{O}(n+m)$.

Proof. Clearly the theorem is true for $\delta = 1$, so assume that $\delta \geq 2$. Let T_δ be the vertices added to T in line 5 of the Heuristic TOTAL DOMINATION with $d_{\text{Not\_TD}}(x) \geq \delta$ (see line 4 of the heuristic). Let T_i be the vertices added to T with $d_{\text{Not\_TD}}(x) = i$ for $i = 1,2,\ldots,\delta-1$. Let Not_TD_i denote the set Not_TD after having added all vertices in $T_{i+1} \cup T_{i+2} \cup \cdots \cup T_\delta$ to T. That is, $\text{Not\_TD}_i = V(G) \setminus (\cup_{j=i+1}^{\delta} N(T_j))$. Furthermore let $t_i = |T_i|$ for all $i = 1,2,\ldots,\delta$. Then,

$$T = \bigcup_{i=1}^{\delta} T_i \quad \text{and} \quad |T| = \sum_{i=1}^{\delta} t_i.$$

Given any $i \in \{1,2,\ldots,\delta\}$, we have that

$$\sum_{v\in \text{Not\_TD}_i} d_G(v) = \sum_{w\in V(G)} d_{\text{Not\_TD}_i}(w), \tag{3.1}$$

since every edge with exactly one end in Not_TD_i adds 1 to both sides of the equation, every edge with both ends in Not_TD_i adds 2 to both sides of the equation, and all other edges do not affect either side. Further since $d_{\text{Not\_TD}_i}(w) \leq i$ for all $w \in V(G)$, Eq. (3.1) implies that

$$\sum_{v \in \text{Not\_TD}_i} d_G(v) \leq in. \tag{3.2}$$

For $1 \leq j \leq i \leq \delta - 1$, every vertex $x \in T_j$ removes j vertices from Not_TD_j, while every vertex $x \in T_\delta$ removes at least δ vertices from Not_TD_δ. Hence, for each value of i with $i \in \{1, 2, \ldots, \delta\}$, we have that

$$|\text{Not\_TD}_i| = \sum_{j=1}^{i} \left(\sum_{x \in T_j} d_{\text{Not\_TD}_j}(x) \right) \geq \sum_{j=1}^{i} jt_j. \tag{3.3}$$

By Eqs. (3.2) and (3.3), we therefore have that for each value of i with $i \in \{1, 2, \ldots, \delta\}$,

$$\delta \sum_{j=1}^{i} jt_j \leq \delta |\text{Not\_TD}_i| \leq \sum_{v \in \text{Not\_TD}_i} d_G(v) \leq in,$$

and so,

$$\sum_{j=1}^{i} jt_j \leq \frac{in}{\delta}.$$

Therefore there exist nonnegative real numbers $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_\delta$ such that

$$\sum_{j=1}^{i} jt_j = \frac{in}{\delta} - \varepsilon_i$$

for all $i = 0, 1, \ldots, \delta$, where $\varepsilon_0 = 0$. This implies that for each value of j with $j \in \{1, 2, \ldots, \delta\}$, we have the following equation:

$$jt_j = \sum_{i=1}^{j} it_i - \sum_{i=1}^{j-1} it_i = \frac{jn}{\delta} - \varepsilon_j - \left[\frac{(j-1)n}{\delta} - \varepsilon_{j-1} \right] = \frac{n}{\delta} - (\varepsilon_j - \varepsilon_{j-1}).$$

Thus,

$$|T| = \sum_{i=1}^{\delta} t_i = \sum_{i=1}^{\delta} \left(\frac{n}{i\delta} \right) - \frac{\varepsilon_\delta}{\delta} - \sum_{i=1}^{\delta-1} \varepsilon_i \left(\frac{1}{i} - \frac{1}{i+1} \right) \leq \sum_{i=1}^{\delta} \left(\frac{n}{i\delta} \right).$$

Therefore,

$$|T| \leq \frac{n}{\delta} \sum_{i=1}^{\delta} \frac{1}{i} \leq \frac{n}{\delta} (1 + \ln \delta) = \left(\frac{1 + \ln \delta}{\delta} \right) n.$$

This completes the proof of the bound in the theorem. The time complexity follows from the discussion preceding the statement of the theorem. □

For sufficiently large δ, the bound on the size of the TD-set produced by Heuristic TOTAL DOMINATION is close to optimal, as can be deduced from the following result.

Theorem 3.4 ([207]). *For every $k \geq 1$, there exists a bipartite k-regular graph, G, with $\gamma_t(G) > \left(\frac{0.1\ln(k)}{k}\right) n(G)$.*