Exact Algorithms for the Connected Dominating Set Problem in Undirected Graphs (Postprint)

Authors: Zhou Xiaoqing, Ye Ansheng, Zhang Zhiqiang

Date: 2018-05-24T00:00:00+00:00

Abstract

A dominating set D V of a graph G=(V,E) is a vertex subset such that every vertex in the graph is either in D or adjacent to at least one vertex in D. The Connected Dominating Set problem is to find a dominating set S with the minimum number of vertices such that the induced subgraph G[S] is connected. This problem is a classic NP-hard problem with applications in connected facility location, ad-hoc networks, and other fields. For the Connected Dominating Set problem in undirected graphs, we carefully analyze the graph structural properties of the problem, derive several effective reduction rules and branching rules, design a branch-and-search algorithm, and employ the Measure and Conquer method to analyze the algorithm' s running time, finally obtaining an exact algorithm with a running time complexity of O*(1.93^n).

Full Text

Preamble

Exact Algorithms for the Connected Dominating Set Problem in Undirected Graphs

Zhou Xiaoqing[1,2], Ye Ansheng[2]†, Zhang Zhiqiang[2]

(1. School of Computer Science & Engineering, University of Electronic Science & Technology of China, Chengdu 611731, China;
2. College of Information Science & Engineering, Chengdu University, Chengdu 610106, China)

Abstract: A dominating set of a graph is a subset of vertices such that every vertex of the graph is either in the set or adjacent to at least one vertex in the set. The connected dominating set problem asks to find a dominating set with the minimum number of vertices whose induced subgraph is connected. This problem is a classical NP-hard problem with applications in connected facility

location, ad-hoc networks, and other areas. For the connected dominating set problem in undirected graphs, this paper carefully analyzes structural properties, explores several effective reduction rules and branching rules, and designs a branch-and-search algorithm. Using the measure-and-conquer method to analyze the running time, we obtain an exact algorithm with time complexity $\mathcal{O}^*(1.9378^n)$.

Keywords: NP-hard problem; exact algorithm; measure-and-conquer; connected dominating set problem

0 Introduction

The history of nontrivial exact algorithms dates back to 1962 when Held and Karp [**?**] designed a dynamic programming algorithm for the Traveling Salesman Problem (TSP) with running time $\mathcal{O}^*(2^n)$ (where n is the number of vertices in the graph), which remains one of the fastest algorithms for this problem to date. Whether there exists an algorithm for TSP with running time $\mathcal{O}^*(1.99^n)$ has become a famous open problem in computational theory. Another well-known problem is the Maximum Independent Set problem. In 1977, Tarjan et al. [**?**] proposed an algorithm with running time $\mathcal{O}^*(2^{n/3})$. After research by several scholars, Robson [**?**] improved the upper bound to $\mathcal{O}^*(1.1996^n)$ in 1986. However, improving this bound below 1.2^n took more than 30 years.

For different NP-hard problems, the time complexities of currently known exact algorithms vary significantly, as exemplified by the TSP and Maximum Independent Set problems mentioned above. This raises important questions: Are there connections between problems with poor time complexity? Does solving one NP-hard problem help solve others? Can we classify classic NP-complete problems to predict how good the worst-case time complexity can be? These considerations help us better understand the computational complexity of NP-complete problems.

The concept of domination in graphs was first described by Claude Berge [**?**] in 1962. In the same year, Ore [**?**] first used the term "dominating set" in the literature. Dominating sets include vertex dominating sets and edge dominating sets, and the dominating set problem includes both vertex and edge domination variants. The Minimum Vertex Dominating Set problem is a classic NP-hard problem [**?**]. In 2004, Fomin et al. [**?**] designed an exact algorithm with running time $\mathcal{O}^*(1.9378^n)$, first breaking the trivial $\mathcal{O}^*(2^n)$ bound. The currently best exact algorithm for this problem runs in $\mathcal{O}^*(1.4969^n)$ time [**?**]. For the Minimum Edge Dominating Set problem, Yannakakis and Gavril [**?**] proved its NP-hardness. Schiermeyer [**?**] first broke the $\mathcal{O}^*(2^n)$ bound in 2008, and the currently best exact algorithm runs in $\mathcal{O}^*(1.3160^n)$ time [**?**].

For the Connected Dominating Set problem studied in this paper, Garey and Johnson [**?**] proved it to be NP-hard. In 2008, Fomin et al. [**?**] first broke the

$\mathcal{O}^*(2^n)$ bound, proposing an exact algorithm with running time $\mathcal{O}^*(1.9407^n)$. At first glance, the Connected Dominating Set problem appears closely related to the Dominating Set problem, suggesting that methods for the latter might solve the former. However, this is not feasible because, from the perspective of exact algorithms, these problems differ significantly. In fact, the Connected Dominating Set problem belongs to a class called "non-local problems," which are typically difficult to handle. The classic TSP problem is another example. Since exact algorithms for Dominating Set are usually based on local structures and cannot effectively capture the global connectivity property, it is difficult to adapt Dominating Set techniques to solve Connected Dominating Set. Other similar non-local problems include the Feedback Vertex Set problem and the Steiner Tree problem.

This paper presents an exact algorithm for the Connected Dominating Set problem with running time $\mathcal{O}^*(1.9378^n)$. The basic algorithmic idea follows Fomin et al.' s [**?**] approach of "maintaining connectivity," where partial solutions generated during recursion must remain connected, combined with the measure-and-conquer method [**?**, **?**] for analysis. When analyzing the time complexity of recursive algorithms, we employ a straightforward metric setting and clear complexity analysis (see Section 4). We carefully analyze the structural properties of the Connected Dominating Set problem, identify several effective reduction and branching rules that simplify the problem, and design a branch-and-search algorithm. The properties and theorems developed can reduce problem size and difficulty in practical applications, making them suitable for combination with heuristic algorithms.

---

1 Notation and Problem Definition

A simple graph is an undirected graph without parallel edges or loops. All graphs discussed in this paper are undirected simple graphs. Let $G = (V, E)$ denote a graph with vertex set V and edge set E, where $|V| = n$ and $|E| = m$. If a vertex subset contains only one element $\{v\}$, we simply denote it as v. For any vertex v in graph G, $N(v)$ denotes the open neighborhood of v, $N[v]$ denotes the closed neighborhood, and $N^2(v)$ denotes the set of vertices at distance 2 from v. The degree of vertex v is denoted by $d(v)$, which counts edges incident to v. In simple graphs, $d(v) = |N(v)|$ always holds. For a vertex subset X, let $N(X) = \bigcup_{v \in X} N(v) \setminus X$ denote the neighbors of X outside X. The subgraph induced by vertex subset X is denoted by $G[X]$, which can be abbreviated as $[X]$ when G is clear from context. If the induced subgraph $G[X]$ is connected, we call X a connected subset.

For algorithm runtime analysis, we use $\mathcal{O}^*$ notation to suppress polynomial factors. For two functions $f(n)$ and $g(n)$, $f(n) = \mathcal{O}^*(g(n))$ means $f(n) = \mathcal{O}(g(n) \cdot \text{poly}(n))$.

Without loss of generality, we make the following assumptions: all graphs men-

tioned are connected, otherwise the problem has no solution; the size of a minimum connected dominating set in G is at least 2, otherwise the problem can be solved in polynomial time.

During algorithm execution, some vertices must be included in the final solution. Let S denote the set of these vertices, where the induced subgraph $G[S]$ is connected. Simultaneously, some vertices must not be in the final solution, denoted by set D, which are deleted from the graph during execution. Let OPT denote an optimal solution. Then all vertices in S must be in OPT, and all vertices in D must not be in OPT. Let $A = V \setminus (S \cup D)$ denote vertices that have not been processed, called the available vertex set. If deleting a vertex $v \in A$ from the graph makes the problem instance unsolvable, we call v a mandatory vertex. If a vertex $v \in A$ is adjacent to at least one vertex in S, we call v a candidate vertex; otherwise, it is a free vertex. Let C denote the set of candidate vertices, $C = N(S) \cap A$. Let F denote the set of free vertices, $F = A \setminus C$.

If a vertex v in graph G is adjacent to at least one vertex in set S, we say that S dominates v. This paper considers the following constrained minimum connected dominating set problem:

Constrained Minimum Connected Dominating Set Problem

Input: An undirected graph $G = (V, E)$ and two vertex subsets $S, D \subseteq V$ where the induced subgraph $G[S]$ is connected.

Problem: Find a minimum vertex subset Y of G satisfying: $S \subseteq Y$, the induced subgraph $G[Y]$ is connected, and Y dominates all vertices in V.

When S contains any two adjacent vertices u and v, the constrained minimum connected dominating set problem becomes finding a minimum connected dominating set containing u and v. We study this constrained version because, during branch-and-search algorithms, some branches require keeping certain vertices in the final solution, which can be directly added to S. For the constrained minimum connected dominating set problem, any vertex subset containing all vertices in S and forming a connected dominating set is called a feasible solution. When a feasible solution has minimum size, it is called an optimal solution, which we abbreviate as "a solution" in this paper.

---

2 Algorithm Design

2.1 Preliminary Ideas

Due to the global connectivity property of the minimum connected dominating set problem, our initial algorithmic approach is to guess that any two adjacent vertices v_1 and v_2 in the graph might appear in some optimal solution. Then we select a candidate vertex v from C and consider two cases: either v is in the solution or not. If v is in the solution, we add v to S; if not, we delete

v from the graph. This branching search continues until C becomes empty or contains no candidate vertices. While this approach solves the problem, its theoretical analysis only yields a running time bound of $\mathcal{O}^*(2^n)$, which remained unimproved for a long time until Fomin et al. [**?**] applied the measure-and-conquer method.

When solving the connected dominating set problem, Fomin et al.' s main idea was to continuously select a vertex v and consider two cases: either keep v in the final solution or delete it from the graph. However, when adding a vertex to the solution, they ensured the solution "remains connected." This paper follows the same "connectivity-maintaining" principle but provides more refined treatment of candidate vertices in C. When considering adding a candidate vertex $v \in C$ to the solution, there are two purposes: (1) to dominate all free-point neighbors of v, or (2) to connect some free-point neighbor of v. In the first case, after adding v to the solution, we can simply delete all its free-point neighbors. In the second case, after adding v, we can add one of its free-point neighbors to the solution. Based on this idea, we continuously process candidate vertices in C. When $C = \emptyset$ or no candidate vertices exist, the algorithm either finds a solution or determines that none exists. Finally, we use the measure-and-conquer method to analyze and design the algorithm, further improving its runtime bound.

2.2 Theorems and Properties

If a solution to instance I can be constructed from a solution to another instance I' in polynomial time, we say problem instance I is equivalent to I', and vice versa. We first prove an equivalence property for problem instances, then define reduction operations based on this property to simplify the original problem instance.

Theorem 1. For a constrained minimum connected dominating set problem instance $I = (G, S, D)$, if I has a solution that does not contain some vertex $v \in A$, then any solution of the instance $I' = (G \setminus v, S, D)$ obtained by deleting v is also a solution of the original problem instance I. Conversely, if I has a solution containing some vertex $v \in A$, then any solution of the instance $I'' = (G, S \cup \{v\}, D)$ obtained by adding v to S is also a solution of the original problem instance I.

Proof: Let OPT be any optimal solution of I. Since $G \setminus v$ is a subgraph of G, OPT is also a feasible solution for I'. Suppose an optimal solution OPT' of I' does not contain vertex v. Then OPT' remains a feasible solution for I. In this case, $|OPT'| \leq |OPT|$, so OPT' is an optimal solution for I.

Let OPT'' be any optimal solution of I''. Since the induced subgraph $G[OPT'']$ is connected, OPT'' is a feasible solution for I. Suppose an optimal solution of I contains vertex v. Then OPT'' remains a feasible solution for I. In this case, $|OPT''| \leq |OPT|$, so OPT'' is an optimal solution for I.

From Theorem 1, if we know a vertex is in an optimal solution, we can add it

to S; if we know a vertex is not in an optimal solution, we can delete it directly.

Property 1. For a constrained minimum connected dominating set problem instance $I = (G, S, D)$, if there exists a candidate vertex $v \in C$ that is mandatory, then add vertex v to S.

Since v is mandatory, deleting v makes the problem instance unsolvable, so v must be in the final solution. By Theorem 1, Property 1 follows directly.

Property 2. If there are two candidate vertices v and u, and by Property 1 neither is mandatory, suppose $N(v) \cap F \subseteq N(u) \cap F$. Then delete vertex v from the graph.

Assume an optimal solution OPT contains vertex v. Since all free points dominated by v are also dominated by u, and both v and u are already dominated, there exists another optimal solution $OPT' = OPT \setminus \{v\} \cup \{u\}$ with $|OPT'| \leq |OPT|$. Thus, v can be deleted.

Property 3. If there exists an available vertex $v \in A$ that does not dominate any free points, then add vertex v to S.

Since v does not dominate any free points, all neighbors of v are candidate points, and all candidate points are connected to v. Therefore, removing v from any feasible solution maintains feasibility.

After reduction by Property 3, we can easily see that if the graph contains a candidate point v, then v must be adjacent to at least one free point; otherwise, we could directly add v to S.

Property 4. Suppose graph G contains a candidate point v that dominates an available point u. If after adding v to S, u no longer dominates any free points, then add u to S.

Because after adding candidate v to S, available point u no longer dominates any free points. By Property 3, when an available point no longer dominates any free points, we can add it to S.

After reduction by Property 4, we can easily see that if a candidate point v dominates an available point u, then v must dominate another free point.

Property 5. Suppose graph G contains a candidate point v that dominates only one free point w, i.e., $N(v) \cap F = \{w\}$. If instance I has an optimal solution containing v but not w, then add w to S.

Proof: Let OPT be an optimal solution of I. Assume OPT contains v but not w. Since OPT must dominate w, at least one neighbor of w is in OPT. If $v \in OPT$, the property holds directly. If $v \notin OPT$, by Property 5 we can directly add w to S.

Property 6. Suppose graph G contains a candidate point v that dominates only one free point w, and w also dominates only one free point u. If instance I

has an optimal solution containing v but not w, then either add w to S or add u to S.

Proof: Let OPT be an optimal solution of I. Assume OPT contains v but not w. To derive a contradiction, suppose OPT does not contain u either. Then there exists another feasible solution $OPT' = OPT \setminus \{v\} \cup \{u\}$ (where u is some appropriate vertex). Since OPT' remains connected and v only dominates w, while w is dominated by u, OPT' is a connected dominating set of size $|OPT'| = |OPT| - 1$, contradicting the optimality of OPT. See case 1 in [Figure 1: see original paper].

[Figure 1: see original paper] Cases where candidate point v dominates only one free point w in graph G

Theorem 2. Given an instance $I = (G, S, D)$, suppose graph G contains a candidate point v that dominates only one free point w, and instance I has an optimal solution OPT such that:

a) If w dominates only one free point u, i.e., $N(w) \cap F = \{u\}$, then either OPT contains w or there exists another optimal solution OPT' that does not contain w.

b) If w dominates at least two free points, let $N(w) \cap F = \{u_1, u_2, \dots, u_k\}$ where $k \geq 2$, then either OPT contains w or OPT contains some u_i where $i \in \{1, 2, \dots, k\}$.

Proof:

a) Let OPT be an optimal solution. Assume OPT contains v; otherwise, a) holds directly. OPT must not contain w, otherwise there would exist a feasible solution $OPT' = OPT \setminus \{w\}$ with $|OPT'| < |OPT|$, contradicting optimality. Since OPT is a dominating set, it must dominate w, so at least one neighbor of w is in OPT. If $u \in OPT$, a) holds directly. If $u \notin OPT$, by Property 6 we can find another optimal solution not containing w, so a) holds.

b) Let OPT be an optimal solution. Assume OPT contains v; otherwise, b) holds directly. OPT must not contain w, otherwise there would exist a feasible solution $OPT' = OPT \setminus \{w\}$ with $|OPT'| < |OPT|$, contradicting optimality. Since OPT must dominate w, at least one neighbor of w is in OPT. If some $u_i \in OPT$, b) holds directly. If no $u_i \in OPT$, by Property 5 we can directly add w to S, so b) holds.

After reduction by Theorem 2, we can easily see that if the graph contains a candidate point v, then v must be adjacent to at least two free points.

Property 7. Suppose graph G contains a candidate point v that dominates exactly two free points w_1 and w_2. If instance I has an optimal solution OPT containing v but not w_1, and w_2 also dominates only one free point u, then either add w_1 to S or add u to S.

Proof: Let OPT be an optimal solution. Assume OPT contains v but not w_1. To derive a contradiction, suppose OPT does not contain u either. Then there exists another feasible solution $OPT' = OPT \setminus \{v\} \cup \{u\}$ (where u is some appropriate vertex). Since OPT' remains connected, v only dominates w_1 and w_2, and w_1 is dominated by u, OPT' is a connected dominating set of size $|OPT'| = |OPT| - 1$, contradicting the optimality of OPT.

[Figure 2: see original paper] Cases where candidate point v dominates two free points w_1 and w_2 in graph G

Property 8. Suppose graph G contains a candidate point v that dominates exactly two free points w_1 and w_2, where w_1 and w_2 each dominate only one free point, u_1 and u_2 respectively. If instance I has an optimal solution OPT containing v but not w_1 and w_2, then either OPT does not contain u_1 and u_2, or there exists another optimal solution OPT' that does not contain v.

Proof: Let OPT be an optimal solution. Assume OPT contains v but not w_1 and w_2. To derive a contradiction, suppose OPT contains u_1 or u_2. Since OPT must dominate w_1 and w_2, at least one neighbor of each is in OPT. If $u_1 \in OPT$ or $u_2 \in OPT$, we can replace v in OPT with u_1 or u_2 to obtain another feasible solution OPT' with $|OPT'| = |OPT|$, so v can be discarded. See case 4 in [Figure 2: see original paper].

Theorem 3. Given an instance $I = (G, S, D)$, suppose graph G contains a candidate point v that dominates exactly two free points w_1 and w_2, and instance I has an optimal solution OPT such that:

a) If w_1 dominates only one free point u_1 and w_2 dominates only one free point u_2, then either OPT contains w_1 or w_2, or OPT contains some u_i where $i \in \{1, 2\}$.

b) If w_1 dominates only one free point u_1 and w_2 dominates at least two free points, let $N(w_2) \cap F = \{u_2\} \cup U$ where $U \neq \emptyset$, then either OPT contains w_1 or w_2, or OPT contains some u_i where $i \in \{1, 2\}$.

Proof:

a) Let OPT be an optimal solution. Assume OPT contains v; otherwise, a) holds directly. If w_1 and w_2 are adjacent, we can replace v with w_1 in OPT to obtain a feasible solution OPT' without v, with $|OPT'| = |OPT|$. Since this case is already considered in the branch that discards v, we need not consider w_1 and w_2 being adjacent. If w_1 and w_2 jointly dominate a free point, we can replace v with w_1 in OPT to obtain OPT' without v. This case is also already considered. Therefore, we consider the case where w_1 and w_2 dominate different free points u_1 and u_2. If neither u_1 nor u_2 is in OPT, by Property 8, either the optimal solution excludes both w_1 and w_2, or there exists another optimal solution excluding v, thus a) holds. If both u_1 and u_2 are in OPT, there exists a feasible solution $OPT' = OPT \setminus \{v\}$ with $|OPT'| < |OPT|$, contradicting optimality, so

u_1 and u_2 cannot both be in OPT. If $u_1 \in OPT$, a) holds directly. If $u_2 \in OPT$, we must discard v, and a) also holds.

b) Let OPT be an optimal solution. Assume OPT contains v; otherwise, b) holds directly. If w_1 and w_2 are adjacent, we can replace v with w_1 in OPT to obtain OPT' without v, with $|OPT'| = |OPT|$. This case is already considered. If w_1 and w_2 jointly dominate a free point, we can replace v with w_1 in OPT to obtain OPT' without v. This case is also already considered. Therefore, we consider the case where w_1 and w_2 dominate distinct free points and each dominates at least two free points. Let $N(w_2) \cap F = \{u_2\} \cup U$. If neither u_1 nor u_2 is in OPT, by Property 7, either add w_1 to S or add w_2 to S, thus b) holds. If both u_1 and u_2 are in OPT, there exists a feasible solution $OPT' = OPT \setminus \{v\}$ with $|OPT'| < |OPT|$, contradicting optimality, so u_1 and u_2 cannot both be in OPT. If $u_1 \in OPT$, b) holds directly. If $u_2 \in OPT$, we must discard v, and b) also holds.

After reduction by Theorem 3, we can easily see that if the graph contains a candidate point v, then v must be adjacent to at least three free points.

2.3 Algorithm Design

Based on the above theorems and properties, we now present the detailed algorithm design, whose main steps are shown in Algorithm 1.

Algorithm 1. MCD(G, S, D)

Input: An undirected graph $G = (V, E)$ and two vertex subsets $S, D \subseteq V$ where $G[S]$ is connected.

Output: A minimum connected dominating set Y of G satisfying $S \subseteq Y$.

1. If $G[S]$ is not connected, stop the algorithm and return $\emptyset$.
2. Else if $G[S]$ is connected, return S as the solution.
3. Else if $A = \emptyset$ and no candidate vertices exist, return $\emptyset$.
4. Else if there exists a mandatory candidate vertex v in the graph, return MCD$(G, S \cup \{v\}, D)$.
5. Else if there exist two candidate vertices v and u (neither mandatory) and $N(v) \cap F \subseteq N(u) \cap F$, return MCD$(G, S, D \cup \{v\})$.
6. Else if there exists an available vertex v that does not dominate any free points (i.e., all neighbors of v are dominated), return MCD$(G, S \cup \{v\}, D)$.
7. Else if there exists a candidate vertex v that dominates an available vertex u, where after selecting v, u no longer dominates any free points, return the better solution between MCD$(G, S, D \cup \{v\})$ and MCD$(G, S \cup \{v\}, D \cup \{u\})$.
8. Else if there exists a candidate vertex v that dominates only one free point w, and w also dominates only one free point, return the better solution between MCD$(G, S, D \cup \{v\})$ and MCD$(G, S \cup \{v, w\}, D)$.
9. Else if there exists a candidate vertex v that dominates only one free point w, and w dominates at least two free points, let $N(w) \cap F = \{u_1, u_2, \ldots, u_k\}$

where $k \geq 2$, return the best solution among $\mathrm{MCD}(G, S, D \cup \{v\})$, $\mathrm{MCD}(G, S \cup \{v, w\}, D)$, and $\mathrm{MCD}(G, S \cup \{v\}, D \cup \{w\})$.

10. Else if there exists a candidate vertex v that dominates two free points w_1 and w_2, where w_1 and w_2 each dominate only one free point u_1 and u_2 respectively, return the best solution among $\mathrm{MCD}(G, S, D \cup \{v\})$, $\mathrm{MCD}(G, S \cup \{v, w_1\}, D)$, $\mathrm{MCD}(G, S \cup \{v, w_2\}, D \cup \{w_1\})$, and $\mathrm{MCD}(G, S \cup \{v\}, D \cup \{w_1, w_2\})$.
11. Else if there exists a candidate vertex v that dominates two free points w_1 and w_2, where w_1 and w_2 each dominate at least two free points, let $N(w_1) \cap F = \{u_1\} \cup U_1$ and $N(w_2) \cap F = \{u_2\} \cup U_2$ where $U_1, U_2 \neq \emptyset$, return the best solution among $\mathrm{MCD}(G, S, D \cup \{v\})$, $\mathrm{MCD}(G, S \cup \{v, w_1\}, D)$, $\mathrm{MCD}(G, S \cup \{v, w_2\}, D \cup \{w_1\})$, $\mathrm{MCD}(G, S \cup \{v\}, D \cup \{w_1\})$, and $\mathrm{MCD}(G, S \cup \{v\}, D \cup \{w_2\})$.
12. Else (there exists a candidate vertex v with at least three free-point neighbors), return the better solution between $\mathrm{MCD}(G, S, D \cup \{v\})$ and $\mathrm{MCD}(G, S \cup \{v\}, D)$.

The algorithm consists of 12 steps described recursively, where each step calls the algorithm itself. Steps 1, 2, and 3 handle base cases and are clearly correct. Step 4 adds mandatory points to S, with correctness given by Property 1. Step 5 deletes a candidate vertex v when all free points it dominates are also dominated by another candidate u, with correctness given by Property 2. Step 6 deletes available points that don' t dominate any free points, with correctness from Property 3. Step 7 handles a candidate v dominating an available point u where u becomes non-dominating after selecting v. It branches into: (1) deleting v, or (2) adding v to S and deleting u. Correctness follows from Property 4.

Step 8 handles a candidate v dominating a single free point w, where w also dominates only one free point. It branches into: (1) deleting v, or (2) adding both v and w to S. Correctness follows from Theorem 2. Step 9 handles a candidate v dominating one free point w, where w dominates at least two free points. Let $N(w) \cap F = \{u_1, u_2, \ldots, u_k\}$. It branches into three subcases: (1) delete v, (2) add v and w to S, or (3) add v to S while deleting w. Correctness follows from Theorem 2.

Steps 10 and 11 handle a candidate v dominating two free points w_1 and w_2. In Step 10, where w_1 and w_2 each dominate only one free point (u_1 and u_2), the algorithm has four branches: (1) delete v, (2) add v and w_1 to S, (3) add v and w_2 to S while deleting w_1, or (4) add v to S while deleting w_1 and w_2. Correctness follows from Theorem 3. In Step 11, where w_1 and w_2 each dominate at least two free points, let $N(w_1) \cap F = \{u_1\} \cup U_1$ and $N(w_2) \cap F = \{u_2\} \cup U_2$. The algorithm has five branches: (1) delete v, (2) add v and w_1 to S, (3) add v and w_2 to S while deleting w_1, (4) add v to S while deleting w_1, or (5) add v to S while deleting w_2. Correctness follows from Theorem 3.

Step 12 performs simple branching on v: either add v to S or delete it. Since v dominates at least three free points, when v is added to S, at least three free points become candidates, reducing the measure by at least $3(1-\alpha)$. This step

is clearly correct.

As long as $A \neq \emptyset$ in the graph, at least one of steps 3-12 can be executed. When $G[S]$ is connected, step 2 finds the solution; when $G[S]$ is not connected, step 1 stops the algorithm. The above analysis establishes the algorithm' s correctness. We now analyze its running time.

---

3 Runtime Analysis Using Measure-and-Conquer

The above algorithm is a branch-and-search algorithm containing both reduction and branching operations. Steps 1, 2, 3, 4, 5, and 6 perform only reduction operations, while steps 7, 8, 9, 10, 11, and 12 perform branching operations. Since reduction operations do not increase the algorithm' s running time exponentially, we analyze only the branching operations.

When using the measure-and-conquer method to analyze the algorithm' s time complexity, we first define a measure for problem size, then design recurrence relations for all branching operations based on this measure. For an instance (G, S, D) of the constrained minimum connected dominating set problem, we define the measure as:

$$\mu(G, S, D) = |S| + |D| + \alpha \cdot |C| + (|V| - |S| - |D| - |C|)$$

where α is a real number with $0 < \alpha < 1$. The value of α will be determined later by solving the recurrence relations from all branching operations. According to this measure, vertices in S and D have weight 0, candidate vertices have weight α, and all other vertices have weight 1. The sum of all vertex weights in the graph serves as the measure $\mu(G, S, D)$. Clearly, during algorithm execution, the measure value never exceeds the number of vertices in the graph.

Let $T(\mu)$ denote the upper bound on the size of the search tree generated by the algorithm on a problem instance with measure μ. We now derive recurrence relations for each branching operation under this measure.

Step 7: In the first branch, candidate v is deleted, reducing the measure by α. In the second branch, v is added to S and u is deleted, reducing the measure by $1 - \alpha$. Since u dominates at least one free point that is also dominated by v, when v is added to S, the free points dominated by u become candidates, reducing the measure by at least $1 - \alpha$. Thus, the second branch reduces the measure by at least $2 - \alpha$. This yields the recurrence:

$$T(\mu) \leq T(\mu - \alpha) + T(\mu - (2 - \alpha))$$

Step 8: In the first branch, candidate v is deleted, reducing the measure by α. In the second branch, v is added to S and w is also added to the solution,

reducing the measure by $1+\alpha$. The neighbor u of w becomes a candidate, reducing the measure by $1-\alpha$. Thus, the second branch reduces the measure by $2+\alpha$. This gives:

$$T(\mu) \leq T(\mu-\alpha) + T(\mu-(2+\alpha))$$

Step 9: In the first branch, candidate v is deleted, reducing the measure by α. In the second branch, v is added to S and w is also added to S. Since w dominates at least two free points, when w is added to S, the free points it dominates become candidates, reducing the measure by at least $2(1-\alpha)$. Thus, the second branch reduces the measure by at least $1+\alpha+2(1-\alpha)=3-\alpha$. In the third branch, v is added to S to dominate w, so w is deleted along with its free-point neighbors, reducing the measure by at least $2+\alpha$. This yields:

$$T(\mu) \leq T(\mu-\alpha) + T(\mu-(3-\alpha)) + T(\mu-(2+\alpha))$$

Step 10: In the first branch, candidate v is deleted, reducing the measure by α. In the second branch, v is added to S to connect w_1, so v and w_1 are added to S, reducing the measure by at least $1+\alpha$. In the third branch, v is added to S to connect w_2, so v and w_2 are added to S while deleting w_1, reducing the measure by at least 3. In the fourth branch, v is added to S to dominate w_1 and w_2, so v is added to S while deleting w_1 and w_2, reducing the measure by at least $4-\alpha$. This gives:

$$T(\mu) \leq T(\mu-\alpha) + T(\mu-(1+\alpha)) + T(\mu-3) + T(\mu-(4-\alpha))$$

Step 11: In the first branch, candidate v is deleted, reducing the measure by α. In the second branch, v is added to S to connect w_1, so v and w_1 are added to S, reducing the measure by at least $1+\alpha$. In the third branch, v is added to S to connect w_2, so v and w_2 are added to S while deleting w_1, reducing the measure by at least $4-2\alpha$. In the fourth branch, v is added to S to dominate w_1, so v is added to S while deleting w_1 and its free-point neighbors, reducing the measure by at least $4-\alpha$. In the fifth branch, v is added to S to dominate w_2, so v is added to S while deleting w_2 and its free-point neighbors, reducing the measure by at least $4-\alpha$. This yields:

$$T(\mu) \leq T(\mu-\alpha)+T(\mu-(1+\alpha))+T(\mu-(4-2\alpha))+T(\mu-(4-\alpha))+T(\mu-(4-\alpha))$$

Step 12: This step performs simple branching on v: either add v to S or delete it. Since v dominates at least three free points, when v is added to S, at least three free points become candidates, reducing the measure by at least $3(1-\alpha)$. This gives:

$$T(\mu) \leq T(\mu - \alpha) + T(\mu - (3 - 3\alpha))$$

Treating all these recurrences as constraints, we formulate a quasiconvex program to minimize the maximum branching factor. Using the method from [**?**], solving this quasiconvex program yields $\alpha = 0.8644$ and a maximum branching factor of 1.93. Therefore, the algorithm' s running time is bounded by $\mathcal{O}^*(1.93^n)$. Since initially $\mu \leq n$, the constrained minimum connected dominating set problem can be solved in $\mathcal{O}^*(1.93^n)$ time, giving us:

Theorem 4. The Minimum Connected Dominating Set problem can be solved in $\mathcal{O}^*(1.93^n)$ time.

---

4 Conclusion

This paper addresses the minimum connected dominating set problem in undirected graphs by designing a branch-and-search algorithm with running time $\mathcal{O}^*(1.93^n)$ using the measure-and-conquer method. We first analyze problem properties, design reduction rules to simplify instances, prove important theorems, and finally develop a recursive branch-and-search algorithm. The core idea is maintaining solution connectivity while distinguishing the roles of candidate vertices when added to the solution. The properties and theorems can reduce problem size and difficulty in practical applications, making them suitable for combination with heuristic algorithms to improve solution speed and quality. The measure-and-conquer method is a crucial analytical technique that has improved running time bounds for many important NP-hard problems, and its application to more NP-hard problems is an ongoing research direction.

References

[1] Cook S A. The complexity of theorem-proving procedures [C]// Proc of the 3rd ACM Symposium on the Theory of Computing. New York: ACM Press, 1971: 151-158.

[2] Impagliazzo R, Paturi R, Zane F. Which problems have strongly exponential complexity? [J]. Journal of Computer and System Sciences, 2001, 63 (4): 512-530, 2001.

[3] 路纲, 周明天, 唐勇, 等. 任意图支配集精确算法回顾 [J]. 计算机学报, 2010, 33 (6): 1073-1087. (Lu Gang, Zhou Mingtian, Tang Yong, et al. A survey on exact algorithms for dominating set related problems in arbitrary graphs [J]. Chinese Journal of Computers, 2010, 33 (6): 1073-1087.)

[4] Held M, Karp R M. A dynamic programming approach to sequencing problems [J]. Journal of SIAM, 1962, 10 (1): 196-210.

[5] Tarjan R, Trojanowski A. Finding a maximum independent set [J]. SIAM Journal on Computing, 1977, 6 (3): 537-546.

[6] Robson J M. Algorithms for maximum independent sets [J]. Journal of Algorithms, 1986, 7 (3): 425-440.

[7] Mingyu Xiao, Nagamochi H. Exact algorithms for maximum independent set [C]// Proc of the 24th International Symposium on Algorithms and Computation. Berlin: Springer Publishing Company, 2013: 328-338.

[8] Claude B. The theory of graphs and its applications [J]. Bulletin of Mathematical Biophysics, 1962, 24 (4): 441-443.

[9] Ore O. Theory of Graphs [M]. Providence: American Mathematical Society, 1962.

[10] Garey M R, Johnson D S. Computers and intractability: a guide to the theory of NP-completeness [M]. New York: W. H. Freeman, 1979.

[11] Fomin F V, Kratsch D, Woeginger G J. Exact (exponential) algorithms for the dominating set problem [C]// Proc of the 30th Workshop on Graph Theoretic Concepts in Computer Science. Berlin: Springer, 2004: 245-256.

[12] Rooij J M M V, Bodlaender H L. Exact algorithms for dominating set [J]. Discrete Applied Mathematics, 2011, 159 (17): 2147-2164.

[13] Yannakakis M, Gavril F. Edge dominating sets in graphs [J]. SIAM Journal on Applied Mathematics, 1980, 38 (3): 364-372.

[14] Schiermeyer I. Efficiency in exponential time for domination-type problems [J]. Discrete Applied Mathematics, 2008, 156 (17): 3291-3297.

[15] Xiao M, Nagamochi H. A refined exact algorithm for edge dominating set [J]. Theoretical Computer Science, 2014, 560: 207–216.

[16] Fomin F V, Grandoni F, Kratsch D. Solving connected dominating set faster than 2^n [J]. Algorithmica, 2008, 52 (2): 153-166.

[17] Ugurlu O, Tanir D, Nuri E. A better heuristic for the minimum connected dominating set in ad hoc networks [C]// Proc of IEEE International Conference on Application of Information and Communication Technologies. New York: IEEE, 2017: 1-4.

[18] Baïou M, Barahona F. The dominating set polytope via facility location [M]. Berlin: Springer International Publishing, 2014: 38-49.

[19] Mohanty J P, Mandal C, Reade C, et al. Construction of minimum connected dominating set in wireless sensor networks using pseudo dominating set [J]. Ad Hoc Networks, 2016, 42 (C): 61-73.

[20] 黄庆东, 闫乔乔, 孙晴. 一种改进的 ad hoc 无线网络连通支配集生成方法 [J]. 电子科技大学学报, 2017, 46 (6): 819-824. (Huang Qingdong, Yan Qiaoqiao, Sun Qing. An improved formation method of connected-dominating set in Ad hoc

wireless networks [J]. Journal of University of Electronic Science and Technology of China, 2017, 46 (6): 819-824.)

[21] Touil S, Mahfoudhi, Laouamer L, et al. A novel scheme for selecting minimum connected dominating set in Ad hoc and WSNs [J]. Research Journal of Applied Sciences, 2017, 12: 409-415.

[22] Shukla S, Misra R, Agarwal A. Virtual coordinate system using dominating set for GPS-free adhoc networks [J]. Annals of Telecommunications, 2017, 72 (3-4): 1-10.

[23] Fomin F V, Grandoni F, Kratsch D. A measure & conquer approach for the analysis of exact algorithms [J]. Journal of the ACM, 2009, 56 (5): 1-32.

[24] 周晓清, 肖鸣宇. 无向图中子集反馈顶点集问题的精确算法 [J], 计算机学报, 2018, 41 (3): 493-505. (Zhou Xiaoqing, Xiao Mingyu. Exact Algorithms for the Subset Feedback Vertex Set Problem in Undirected Graphs [J]. Chinese Journal of Computers, 2018, 41 (3): 493-505.)

[25] Eppstein D. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms [J]. ACM Trans on Algorithms, 2006, 2 (4): 492-509.

Note: Figure translations are in progress. See original paper for figures.