Chapter 7

Dominating Set

In this chapter we present another randomized algorithm that demonstrates the power of randomization to break symmetries. We study the problem of finding a small dominating set of the network graph. As it is the case for MIS, an efficient dominating set algorithm can be used as a basic building block to solve a number of problems in distributed computing. For example, whenever we need to partition the network into a small number of local clusters, the computation of a small dominating set usually occurs in some way. A particularly important application of dominating sets is for the construction of an efficient backbone for routing.

Definition 7.1 (Dominating Set)**.** *Given an undirected graph $G = (V, E)$, a* dominating set *is a subset $S \subseteq V$ of its nodes such that for all nodes $v \in V$, either $v \in S$ or a neighbor u of v is in S.*

Remarks:

- It is well-known that computing a dominating set of minimal size is NP-hard. We therefore look for approximation algorithms, that is, algorithms which produce solutions which are optimal up to a certain factor.

- Note that every MIS (cf. Chapter 6) is a dominating set. In general, the size of every MIS can however be larger than the size of an optimal minimum dominating set by a factor of $\Omega(n)$. As an example, connect the centers of two stars by an edge. Every MIS contains all the leaves of at least one of the two stars whereas there is a dominating set of size 2.

All the dominating set algorithms that we study throughout this chapter operate in the following way. We start with $S = \emptyset$ and add nodes to S until S is a dominating set. To simplify presentation, we color nodes according to their state during the execution of an algorithm. We call nodes in S *black*, nodes which are covered (neighbors of nodes in S) *gray*, and all uncovered nodes *white*. By $W(v)$, we denote the set of white nodes among the direct neighbors of v, including v itself. We call $w(v) = |W(v)|$ the *span* of v.

7.1 Sequential Greedy Algorithm

Intuitively, to end up with a small dominating set S, nodes in S need to cover as many neighbors as possible. It is therefore natural to add nodes v with a large span $w(v)$ to S. This idea leads to a simple greedy algorithm:

Algorithm 33 Greedy Algorithm

1: $S := \emptyset$;
2: **while** $\exists$ white nodes **do**
3: choose $v \in \{x \mid w(x) = \max_{u \in V}\{w(u)\}\}$;
4: $S := S \cup \{v\}$;
5: **end while**

Theorem 7.2. *The* Greedy Algorithm *computes a* $(\ln \Delta + 2)$*-approximation, that is, for the computed dominating set* S *and an optimal dominating set* S^**, we have*

$$\frac{|S|}{|S^*|} \leq \ln \Delta + 2.$$

Proof. Each time, we choose a new node of the dominating set (each greedy step), we have cost 1. Instead of letting this node pay the whole cost, we distribute the cost equally among all newly covered nodes. Assume that node v, chosen in line 3 of the algorithm, is white itself and that its white neighbors are v_1, v_2, v_3, and v_4. In this case each of the 5 nodes v and $v_1, \ldots, v_4$ get charged $1/5$. If v is chosen as a gray node, only the nodes $v_1, \ldots, v_4$ get charged (they all get $1/4$).

Now, assume that we know an optimal dominating set S^*. By the definition of dominating sets, to each node which is not in S^*, we can assign a neighbor from S^*. By assigning each node to exactly one neighboring node of S^*, the graph is decomposed into stars, each having a dominator (node in S^*) as center and non-dominators as leaves. Clearly, the cost of an optimal dominating set is 1 for each such star. In the following, we show that the amortized cost (distributed costs) of the greedy algorithm is at most $\ln \Delta + 2$ for each star. This suffices to prove the theorem.

Consider a single star with center $v^* \in S^*$ before choosing a new node u in the greedy algorithm. The number of nodes that become dominated when adding u to the dominating set is $w(u)$. Thus, if some white node v in the star of v^* becomes gray or black, it gets charged $1/w(u)$. By the greedy condition, u is a node with maximal span and therefore $w(u) \geq w(v^*)$. Thus, v is charged at most $1/w(v^*)$. After becoming gray, nodes do not get charged any more. Therefore the first node that is covered in the star of v^* gets charged at most $1/(d(v^*)+1)$. Because $w(v^*) \geq d(v^*)$ when the second node is covered, the second node gets charged at most $1/d(v^*)$. In general, the i^{th} node that is covered in the star of v^* gets charged at most $1/(d(v^*)+i-2)$. Thus, the total amortized cost in the star of v^* is at most

$$\frac{1}{d(v^*)+1} + \frac{1}{d(v^*)} + \cdots + \frac{1}{2} + \frac{1}{1} = \mathrm{H}(d(v^*)+1) \leq \mathrm{H}(\Delta+1) < \ln(\Delta) + 2$$

where Δ is the maximal degree of G and where $H(n) = \sum_{i-1}^{n} 1/i$ is the n^{th} number. $\square$

Remarks:

- One can show that unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$, no polynomial-time algorithm can approximate the minimum dominating set problem better than $(1 - o(1)) \cdot \ln \Delta$. Thus, unless P $\approx$ NP, the approximation ratio of the simple greedy algorithm is optimal (up to lower order terms).

7.2 Distributed Greedy Algorithm

For a distributed algorithm, we use the following observation. The span of a node can only be reduced if any of the nodes at distance at most 2 is included in the dominating set. Therefore, if the span of node v is greater than the span of any other node at distance at most 2 from v, the greedy algorithm chooses v before any of the nodes at distance at most 2. This leads to a very simple distributed version of the greedy algorithm. Every node v executes the following algorithm.

Algorithm 34 Distributed Greedy Algorithm (at node v):

while v has white neighbors **do**
    compute span $w(v)$;
    send $w(v)$ to nodes at distance at most 2;
    if $w(v)$ largest within distance 2 (ties are broken by IDs) **then**
        join dominating set
    end if
end while

Theorem 7.3. *Algorithm 34 computes a dominating set of size at most* $\ln \Delta + 2$ *times the size of an optimal dominating set in* $O(n)$ *rounds.*

Proof. The approximation quality follows directly from the above observation and the analysis of the *greedy algorithm*. The time complexity is at most linear because in every iteration of the while loop, at least one node is added to the dominating set and because one iteration of the while loop can be implemented in a constant number of rounds. $\square$

The approximation ratio of the above distributed algorithm is best possible (unless P $\approx$ NP or unless we allow local computations to be exponential). However, the time complexity is very bad. In fact, there really are graphs on which in each iteration of the while loop, only one node is added to the dominating set (even if IDs are chosen randomly). As an example, consider a graph as in Figure 7.1. An optimal dominating set consists of all nodes on the center axis. The *distributed greedy algorithm* computes an optimal dominating set, however, the nodes are chosen sequentially from left to right. Hence, the running time of the algorithm on the graph of Figure 7.1 is $\Omega(\sqrt{n})$. Below, we will see that there are graphs on which Algorithm 34 even needs $\Omega(n)$ rounds.

The problem of the graph of Figure 7.1 is that there is a long path of descending degrees (spans). Every node has to wait for the neighbor to the left. Therefore, we want to change the algorithm in such a way that there are no long

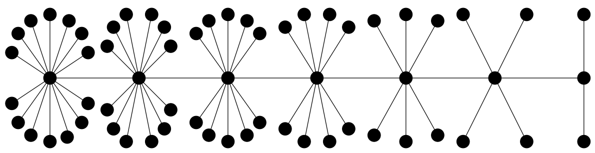

Figure 7.1: Distributed greedy algorithm: Bad example

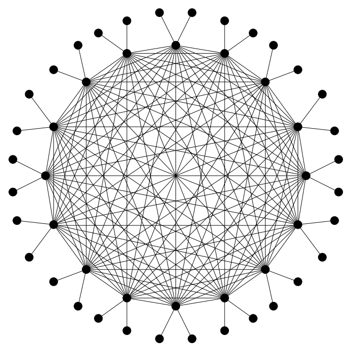

Figure 7.2: Distributed greedy algorithm with rounded spans: Bad example

paths of descending spans. Allowing for an additional factor 2 in the approximation ratio, we can round all spans to the next power of 2 and let the greedy algorithm take a node with a maximal rounded span. In this case, a path of strictly descending rounded spans has length at most $\log n$. For the distributed version, this means that nodes with maximal rounded span within distance 2 are added to the dominating set. Ties are again broken by unique node IDs. If node IDs are chosen at random, the time complexity for the graph of Figure 7.1 is reduced from $\Omega(\sqrt{n})$ to $O(\log n)$.

Unfortunately, there still is a problem remaining. To see this, we consider Figure 7.2. The graph of Figure 7.2 consists of a clique with $n/3$ nodes and two leaves per node of the clique. An optimal dominating set consists of all the $n/3$ nodes of the clique. Because they all have distance 1 from each other, the described distributed algorithm only selects one in each while iteration (the one with the largest ID). Note that as soon as one of the nodes is in the dominating set, the span of all remaining nodes of the clique is 2. They do not have common neighbors and therefore there is no reason not to choose all of them in parallel. However, the time complexity of the simple algorithm is $\Omega(n)$. In order to improve this example, we need an algorithm that can choose many nodes simultaneously as long as these nodes do not interfere too much, even if they are neighbors. In Algorithm 35, $N(v)$ denotes the set of neighbors of v (including v itself) and $N_2(v) = \bigcup_{u \in N(V)} N(u)$ are the nodes at distance at most 2 of v. As before, $W(v) = \{u \in N(v) : u \text{ is white}\}$ and $w(v) = |W(v)|$.
It is clear that if Algorithm 35 terminates, it computes a valid dominating set. We will now show that the computed dominating set is small and that the algorithm terminates quickly.

Algorithm 35 Fast Distributed Dominating Set Algorithm (at node v):

1: $W(v) := N(v)$; $w(v) := |W(v)|$;
2: **while** $W(v) \neq \emptyset$ **do**
3: $\quad \tilde{w}(v) := 2^{\lfloor \log_2 w(v) \rfloor}$; // round down to next power of 2
4: $\quad \hat{w}(v) := \max_{u \in N_2(v)} \tilde{w}(u)$;
5: $\quad$ **if** $\tilde{w}(v) = \hat{w}(v)$ **then** $v.active :=$ **true else** $v.active :=$ **false end if**;
6: $\quad$ compute support $s(v) := |\{u \in N(v) : u.active = \textbf{true}\}|$;
7: $\quad \hat{s}(v) := \max_{u \in W(v)} s(u)$;
8: $\quad v.candidate :=$ **false**;
9: $\quad$ **if** $v.active$ **then**
10: $\qquad v.candidate :=$ **true** with probability $1/\hat{s}(v)$
11: $\quad$ **end if**;
12: $\quad$ compute $c(v) := |\{u \in N(v) : u.candidate = \textbf{true}\}|$;
13: $\quad$ **if** $v.candidate$ **and** $\sum_{u \in W(v)} c(u) \leq 3w(v)$ **then**
14: $\qquad$ node v joins dominating set
15: $\quad$ **end if**
16: $\quad W(v) := \{u \in N(v) : u \text{ is white}\}$; $w(v) := |W(v)|$;
17: **end while**

Theorem 7.4. *Algorithm 35 computes a dominating set of size at most* $(6 \cdot \ln \Delta + 12) \cdot |S^*|$*, where* S^* *is an optimal dominating set.*

Proof. The proof is a bit more involved but analogous to the analysis of the approximation ratio of the greedy algorithm. Every time, we add a new node v to the dominating set, we distribute the cost among v (if it is still white) and its white neighbors. Consider an optimal dominating set S^*. As in the analysis of the greedy algorithm, we partition the graph into stars by assigning every node u not in S^* to a neighbor v^* in S^*. We want to show that the total distributed cost in the star of every $v^* \in S^*$ is at most $6H(\Delta + 1)$.

Consider a node v that is added to the dominating set by Algorithm 35. Let $W(v)$ be the set of white nodes in $N(v)$ when v becomes a dominator. For a node $u \in W(v)$ let $c(u)$ be the number of candidate nodes in $N(u)$. We define $C(v) = \sum_{u \in W(v)} c(u)$. Observe that $C(v) \leq 3w(v)$ because otherwise v would not join the dominating set in line 15. When adding v to the dominating set, every newly covered node $u \in W(v)$ is charged $3/(c(u)w(v))$. This compensates the cost 1 for adding v to the dominating set because

$$\sum_{u \in W(v)} \frac{3}{c(u)w(v)} \;\geq\; w(v) \cdot \frac{3}{w(v) \cdot \sum_{u \in W(v)} c(u)/w(v)} \;=\; \frac{3}{C(v)/w(v)} \;\geq\; 1.$$

The first inequality follows because it can be shown that for $\alpha_i > 0$, $\sum_{i=1}^{k} 1/\alpha_i \geq k/\bar{\alpha}$ where $\bar{\alpha} = \sum_{i=1}^{k} \alpha_i / k$.

Now consider a node $v^* \in S^*$ and assume that a white node $u \in W(v^*)$ turns gray or black in iteration t of the while loop. We have seen that u is charged $3/(c(u)w(v))$ for every node $v \in N(u)$ that joins the dominating set in iteration t. Since a node can only join the dominating set if its span is largest up to a factor of two within two hops, we have $w(v) \geq w(v^*)/2$ for every node $v \in N(u)$ that joins the dominating set in iteration t. Because there are at most $c(u)$ such nodes, the charge of u is at most $6/w(v^*)$. Analogously to the sequential greedy algorithm, we now get that the total cost in the star of a node $v^* \in S^*$ is at most

$$\sum_{i=1}^{|N(v^*)|} \frac{6}{i} \;\le\; 6 \cdot H(|N(v^*)|) \;\le\; 6 \cdot H(\Delta+1) \;=\; 6 \cdot \ln \Delta + 12.$$

□

To bound the time complexity of the algorithm, we first need to prove the following lemma.

Lemma 7.5. *Consider an iteration of the while loop. Assume that a node u is white and that $2s(u) \ge \max_{v \in \mathcal{A}(u)} \hat{s}(v)$ where $\mathcal{A}(u) = \{v \in N(u) : v.active =$ **true**$\}$. Then, the probability that u becomes dominated (turns gray or black) in the considered while loop iteration is larger than $1/9$.*

Proof. Let $D(u)$ be the event that u becomes dominated in the considered while loop iteration, i.e., $D(u)$ is the event that u changes its color from white to gray or black. Thus we need to prove that $\Pr(D(u)) \ge 1/9$. To do so, let $v_1, v_2, \ldots, v_{s(u)}$ be the active nodes in u's neighborhood (with u possibly among them). Define $\mathcal{C}_i$ as the event that node v_i is a candidate and $\mathcal{D}_i$ that node v_i joins the DS. Furthermore let $\mathcal{E}_i$ be the event that v_i becomes a candidate in the current loop, while nodes $v_1, v_2, \ldots, v_{i-1}$ do *not* become candidates, i.e., $\mathcal{E}_i = \mathcal{C}_i \cap \left(\bigcap_{j=1}^{i-1} \overline{\mathcal{C}_j}\right)$; finally, let $\mathcal{E} := \bigcup_{i=1}^{s(u)} \mathcal{E}_i$ be the event that there is at least one candidate in $N(u)$, i.e., that $c(u) > 0$. Note that $\dot{\bigcup}_{i=1}^{s(u)} \mathcal{E}_i \dot{\cup} \overline{\mathcal{E}}$ defines a partition of the probability space Ω, i.e., the events $\overline{\mathcal{E}}$ and $\mathcal{E}_i$ for $i \in \{1, \ldots, s(u)\}$ are disjoint and their union covers all possible events.

For $D(u)$ to happen, at least one of the nodes in $N(u)$ must join the DS and thus we can rewrite $\Pr(D(u))$ as

$$\begin{aligned}
\Pr(D(u)) &\overset{(1)}{=} \sum_{i=1}^{s(u)} \Pr(D(u)|\mathcal{E}_i)\Pr(\mathcal{E}_i) + \underbrace{\Pr(D(u)|\overline{\mathcal{E}})}_{=0}\Pr(\overline{\mathcal{E}}) \\
&\overset{(2)}{\ge} \sum_{i=1}^{s(u)} \Pr(\mathcal{D}_i|\mathcal{E}_i)\Pr(\mathcal{E}_i) \\
&\overset{(3)}{\ge} \sum_{i=1}^{s(u)} \Pr(\mathcal{D}_i|\mathcal{C}_i)\Pr(\mathcal{E}_i).
\end{aligned} \tag{7.1}$$

Equality (1) is an application of the total probability law. Inequality (2) follows because $\mathcal{D}_i \subseteq D(u)$. All nodes decide independently whether to become a candidate. Hence, if we know that some nodes do *not* become candidates, then this can only make it more likely for v_i to pass the test in line 13 of the algorithm, and thus $\Pr(\mathcal{D}_i|\mathcal{E}_i) \ge \Pr(\mathcal{D}_i|\mathcal{C}_i)$, justifying (3).

We claim that $\Pr(\mathcal{D}_i|\mathcal{C}_i) \ge 1/3$ for any i, in which case the inequality above further reduces to

$$\Pr(D(u)) \ge \frac{1}{3}\sum_{i=1}^{s(u)} \Pr(\mathcal{E}_i) \ge \frac{1}{3}\Pr(\mathcal{E}). \tag{7.2}$$

We start with lower bounding $\Pr(\mathcal{E})$. We have $2s(u) \geq \max_{v \in \mathcal{A}(u)} \hat{s}(v)$. Therefore, in line 10, each of the $s(u)$ active nodes $v \in N(u)$ becomes a candidate node with probability $1/\hat{s}(v) \geq 1/(2s(u))$. The probability that at least one of the $s(u)$ active nodes in $N(u)$ becomes a candidate therefore is

$$\Pr(\mathcal{E}) = \Pr(c(u) > 0) > 1 - \left(1 - \frac{1}{2s(u)}\right)^{s(u)} > 1 - \frac{1}{\sqrt{e}} > \frac{1}{3}.$$

We used that for $x \geq 1$, $(1 - 1/x)^x < 1/e$.

We next prove our claim that $\Pr(\mathcal{D}_i|\mathcal{C}_i) \geq 1/3$ for any i. Consider some node v_i and let $C(v_i) = \sum_{v' \in W(v_i)} c(v')$. If v_i is a candidate, it joins the dominating set if $C(v_i) \leq 3w(v_i)$. We are thus interested in the probability $\Pr(C(v_i) \leq 3w(v_i)|\mathcal{C}_i)$. Assume that v_i is a candidate and let $v' \in W(v_i)$ be a white node in the 1-neighborhood of v_i. Let $c'(v') = c(v') - 1$ be the number of candidates in $N(v') \setminus \{v_i\}$. For a node $v' \in W(v_i)$, $c'(v')$ is upper bounded by a binomial random variable $\mathrm{Bin}\big(s(v')-1, 1/s(v')\big)$ with expectation $(s(v') - 1)/s(v')$. We therefore have

$$\mathbb{E}[c(v')|\mathcal{C}_i] = 1 + \mathbb{E}[c'(v')|\mathcal{C}_i] = 1 + \mathbb{E}[c'(v')] = 1 + \frac{s(v') - 1}{s(v')} < 2.$$

By linearity of expectation, we hence obtain

$$\mathbb{E}[C(v_i)|\mathcal{C}_i] = \sum_{v' \in W(v_i)} \mathbb{E}[c(v')|\mathcal{C}_i] < 2w(v_i).$$

We can now use Markov's inequality to bound the probability that $C(v_i)$ becomes too large:

$$\Pr(C(v_i) > 3w(v_i)|\mathcal{C}_i) < \frac{2}{3}.$$

Combining everything, we get

$$Pr(\mathcal{D}_i|\mathcal{C}_i) = \Pr(C(v_i) \leq 3w(v_i)|\mathcal{C}_i) > \frac{1}{3},$$

which, together with (7.2) finishes our proof. □

Theorem 7.6. *In expectation, Algorithm 35 terminates in $O(\log^2\Delta \cdot \log n)$ rounds.*

Proof. First observe that every iteration of the while loop can be executed in a constant number of rounds. Consider the state after t iterations of the while loop. Let $\tilde{w}_{\max}(t) = \max_{v \in V} \tilde{w}(v)$ be the maximal span rounded down to the next power of 2 after t iterations. Further, let $s_{\max}(t)$ be the maximal support $s(v)$ of any node v for which there is a node $u \in N(v)$ with $w(u) \geq \tilde{w}_{\max}(t)$ after t while loop iterations. Observe that all nodes v with $w(v) \geq \tilde{w}_{\max}(t)$ are active in iteration $t+1$ and that as long as the maximal rounded span $\tilde{w}_{\max}(t)$ does not change, $s_{\max}(t)$ can only get smaller with increasing t. Consider the pair $(\tilde{w}_{\max}, s_{\max})$ and define a relation $\prec$ such that $(w', s') \prec (w, s)$ iff $w' < w$ or $w = w'$ and $s' \leq s/2$. From the above observations, it follows that

$$(\tilde{w}_{\max}(t), s_{\max}(t)) \prec (\tilde{w}_{\max}(t'), s_{\max}(t')) \implies t > t'. \tag{7.3}$$

For a given time t, let $T(t)$ be the first time for which

$$(\tilde{w}_{\max}(T(t)), s_{\max}(T(t))) \prec (\tilde{w}_{\max}(t), s_{\max}(t)).$$

We first want to show that for all t,

$$\mathbb{E}[T(t) - t] \;=\; O(\log n). \tag{7.4}$$

Let us look at the state after t while loop iterations Consider a node u that satisfies the following three conditions:

(1) u is white

(2) $\exists v \in N(u) : \tilde{w}(v) = \tilde{w}_{\max}(t)$

(3) $s(u) \geq s_{\max}(t)/2$.

Note that all active neighbors x of u have rounded span $\tilde{w}(x) = \tilde{w}_{\max}$. Thus, for each active neighbor x of u, we have $\hat{s}(x) \leq s_{\max}(t)$. We can therefore apply Lemma 7.5 and conclude that u will be dominated after the following while loop iteration with probability larger than 1/9. Hence, as long as u satisfies all three conditions, the probability that u becomes dominated is larger than 1/9 in every while loop iteration. Hence, after $t+\tau$ iterations (from the beginning), u is dominated or does not satisfy (2) or (3) with probability larger than $(8/9)^\tau$. Choosing $\tau = \log_{9/8}(2n)$, this probability becomes $1/(2n)$. There are at most n nodes u satisfying Conditions $(1)-(3)$. Therefore, applying a union bound, we obtain that with probability more than 1/2, there is no white node u satisfying Conditions $(1)-(3)$ at time $t + \log_{9/8}(2n)$. In that case, either the maximal rounded span is smaller than $\tilde{w}_{\max}$ or the largest span of any node neighboring a node with rounded span $\tilde{w}_{\max}$ is less than $s_{\max}(t)/2$. Therefore, with probability more than 1/2, $T(t) \leq t + \log_{9/8}(2n)$. Analogously, we obtain that with probability more than $1/2^k$, $T(t) \leq t + k\log_{9/8}(2n)$. We then have

$$\begin{aligned}
\mathbb{E}[T(t) - t] &= \sum_{\tau=1}^{\infty} \Pr(T(t) - t = \tau) \cdot \tau \\
&\leq \sum_{k=1}^{\infty} \left(\frac{1}{2^k} - \frac{1}{2^{k+1}}\right) \cdot k \log_{9/8}(2n) \;=\; \log_{9/8}(2n)
\end{aligned}$$

and thus Equation (7.4) holds.

Let $t_0 = 0$ and $t_i = T(t_{i-1})$ for $i = 1, \ldots, k$. where $t_k = \min_t \tilde{w}_{\max}(t) = 0$. Because $\tilde{w}_{\max}(t) = 0$ implies that $w(v) = 0$ for all $v \in V$ and that we therefore have computed a dominating set, by Equations (7.3) and (7.4) (and linearity of expectation), the expected number of rounds until Algorithm 35 terminates is $O(k \cdot \log n)$. Since $\tilde{w}_{\max}(t)$ can only have $\lfloor \log \Delta \rfloor$ different values and because for a fixed value of $\tilde{w}_{\max}(t)$, the number of times $s_{\max}(t)$ can be decreased by a factor of 2 is at most $\log \Delta$ times, we have $k \leq \log^2 \Delta$. $\square$

Remarks:

- It is not hard to show that Algorithm 35 even terminates in $O(\log^2\Delta \cdot \log n)$ rounds w.h.p. (i.e., with probability $1 - 1/n^c$ for an arbitrary constant c).

- Using the median of the supports of the neighbors instead of the maximum in line 8 results in an algorithm with time complexity $O(\log\Delta \cdot \log n)$. With another algorithm, this can even be slightly improved to $O(\log^2\Delta)$.

- One can show that $\Omega(\log\Delta)$ rounds are necessary to obtain an $O(\log\Delta)$-approximation.

- It is not known whether there is a fast deterministic approximation algorithm. This is an interesting and important open problem. The best deterministic algorithm known to achieve an $O(\log\Delta)$-approximation has time complexity $2^{O(\sqrt{\log n})}$.