Design by measure and conquer: A faster exact algorithm for Dominating Set

Johan van Rooij
Utrecht University

Abstract

A dominating set $D \subseteq V$ in a graph $G = (V, E)$ is a subset of the vertices from which we can reach every other vertex by moving along at most one edge, i.e. for every vertex $v \in V$, either $v \in D$ or v has a neighbour $u \in D$. We consider the Dominating Set problem: given a graph G, find a dominating set in G of minimum cardinality. This problem can be seen as a special case of facility location. We solve this problem by modelling it as a Set Cover problem. In such a problem we are given a set of sets $\mathcal{S}$ and asked to compute a subset $\mathcal{C} \subseteq \mathcal{S}$ of minimum cardinality, such that every element in a set in $\mathcal{S}$ is in some set in $\mathcal{C}$.

Both problems are known to be NP-hard. Certain applications require exact solutions to these problems, which motivates the search for fast exponential algorithms solving moderate size instances significantly faster than straightforward enumerative algorithms.

Branch and reduce algorithms are search tree algorithms where reduction rules are applied in each tree node. A recent development in the analysis of these exact branch and reduce algorithms is *measure and conquer*. We use this technique not only for algorithm analysis, but also as a tool in the *design* of algorithms. In an iterative process, we have developed a series of *branch and reduce* algorithms, where each algorithm improves its predecessor. A mathematical analysis of an algorithm in the series with measure and conquer results in a quasiconvex programming problem. The solution by computer to this problem not only gives a bound on the running time, but may also lead to a new reduction rule, thus giving a new, possibly faster algorithm. This makes *design by measure and conquer* a form of *computer aided algorithm design*.

As a result we have obtained the currently fastest known exact algorithms for Dominating Set: an algorithm that uses $O(1.5134^n)$ time and polynomial space, and an algorithm that uses $O(1.5063^n)$ time.