graph searching the objective is to determine the minimal number of guards needed to search a graph. This number is closely related to other important graph parameters, such as interval-width, path-width or vertex separation, see [3,16,35–37]. However, computing the minimal number of searchers is NP-hard [41].

This is the other goal of our approach towards graph automata. So far the graph search strategy has not been of primary concern and has been used only as a means to approach the minimal number of searchers. There are no particular investigations how the searchers do their job, and how the search strategy is described. Graph automata contribute to this point. A graph automaton has only a finite state control and uses only finitely many instructions for the description of a graph search strategy. However, there is some inherent nondeterminism involved. Moreover, there is no a priori upper bound on the number of guards. At last, the search strategy of a graph automaton works only on graphs from the recognized language.

Let's take a closer look at the behaviour of graph automata and the way the guards are directed on an input graph. After each move the guarded nodes separate the input graph into a visited and a yet undiscovered part. The guards watch the edges inbetween, which we shall call bridges. These edges define an edge-separator or a cut of the input graph. Their removal would disconnect the input graph into at least two connected components. In a move, a small piece of the yet unvisited part is discovered, and the frontier of guarded nodes and bridges advances beyond the discovered piece. These moves are continuous, and do not leave a gap. Cleared nodes and edges are not recontaminated. This means a monotone search strategy, graph searching without recontamination [3,4,36,38,41]. Hence, graph automata are plans for monotone search strategies on graphs. The search strategies are special. A strategy is described by a finite set of instructions and is executed by a nondeterministic finite state machine. The behaviour of a graph automaton resembles common graph traversals, such as depth-first search, breadth-first search, or Dijkstra's shortest path algorithm. Such traversals perform a sequence of discrete moves. After each move the graph is partitioned into visited, unvisited and guarded nodes. For efficient graph traversals emphasis is laid on the data structures for the guarded nodes, see [13]. Efficiency is not of concern here.

Graph automata are designed as the dual of important classes of graph grammars. They proceed in such a way that their computations reconstruct derivations of the associated graph grammars, and conversely. This one-to-one correspondence is established for linear and for boundary NCE graph grammars. Our main results state that finite graph automata are equivalent to linear graph grammars and recognize exactly the class of linear NCE graph languages, and that alternating graph automata are equivalent to boundary graph grammars and recognize exactly the class of boundary NCE graph languages.