One-Probe Search

Anna Östlin and Rasmus Pagh
BRICS, University of Aarhus, Denmark

Dagstuhl seminar, February 2002

Arrays

The simplest RAM data structure is the *array*.

An array of size s has the following properties:

- Given an index $i \in \{1, \ldots, s\}$ it returns the word written at position i in the array.
- Uses just one memory access.
- The space usage is s words.

We consider *sparse arrays* where all but $n << s$ entries are empty:

How close can we get to the performance of an array using less space?

Hashing

Hashing in a nutshell:

- Read the description of a hash function h.
- Search for entry i (starting) at memory location $h(i)$.

Properties:

- Uses space $O(n)$.
- Uses at least two memory probes (three in the worst case).
- Good hash functions are hard to maintain deterministically.

What can be done using *one* memory probe?

One-probe sparse arrays

Impossibility:

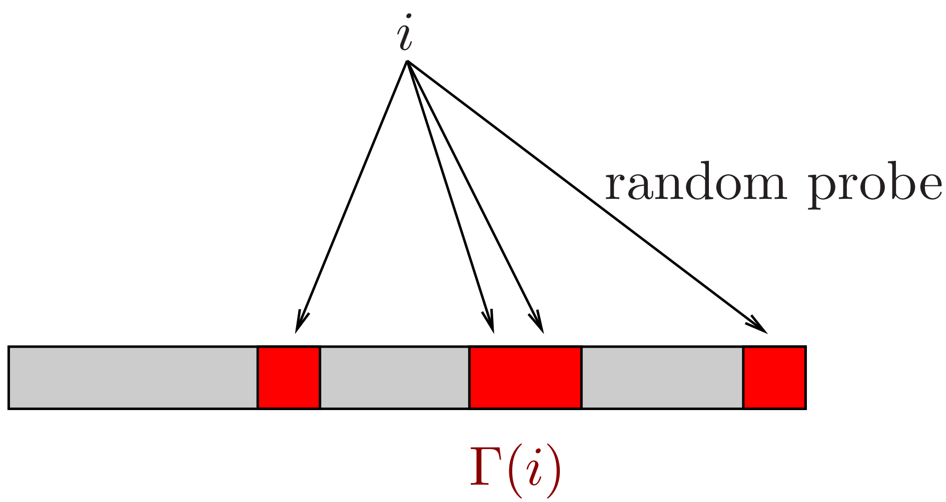
- One-probe search is impossible if the (approximate) size of the data structure is unknown.
- The space usage of an array cannot be improved by any *deterministic* one-probe implementation of a sparse array.

Deterministic lower bounds for membership queries are broken by randomized (Monte Carlo) schemes [Buhrman et al. '00].

Randomized one-probe sparse arrays

Randomized (Las Vegas) one-probe scheme:

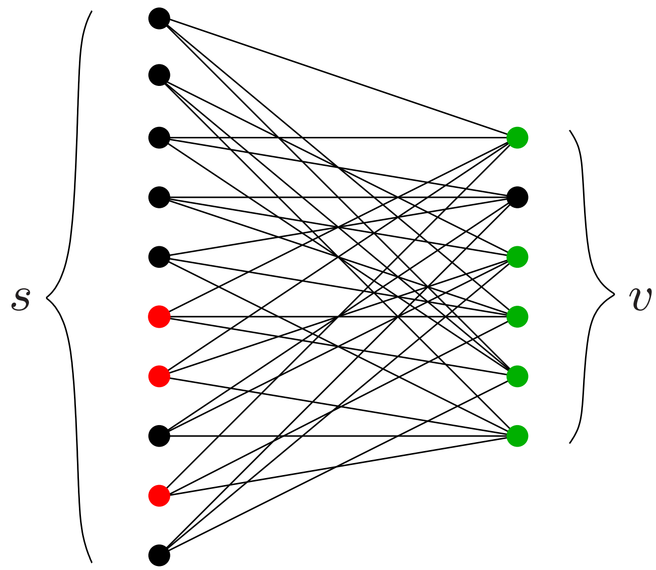
- Knows the size of the data structure.
- Looks up one word chosen at random from the data structure.
- Returns the correct answer with probability $1 - \epsilon$, and otherwise returns *don't know*.
- By iterating, it always returns the correct answer.



Bipartite expander graphs

Bipartite (n, d, ϵ)-expander graph:

Any set of $k \leq n$ nodes (of degree d) on the left have at least $(1-\epsilon)dk$ neighbors.



For any s and constant $\epsilon > 0$ there exists an (n, d, ϵ)-expander graph with $d = O(\log s)$, $v = O(n \log s)$.

Lookup procedure

Suppose we have an $(2n+1, d, \epsilon/2)$-expander with neighbor function $\Gamma : \{1, \ldots, s\} \rightarrow \mathcal{P}(\{1, \ldots, v\})$, $v = O(n \log s)$.

Data structure:

- Array T of length v.
- Each entry has the fields *index* and *info*.

procedure lookup(i)
  choose $v \in \Gamma(i)$ uniformly at random;
  if $T[v]$.index $= i$ **then return** $T[v]$.info
  else if $T[v]$.index $\in \{\neg i, \perp\}$ **then return** *empty*
  else return *don't know*;
end;

Equality-based!

Requirements

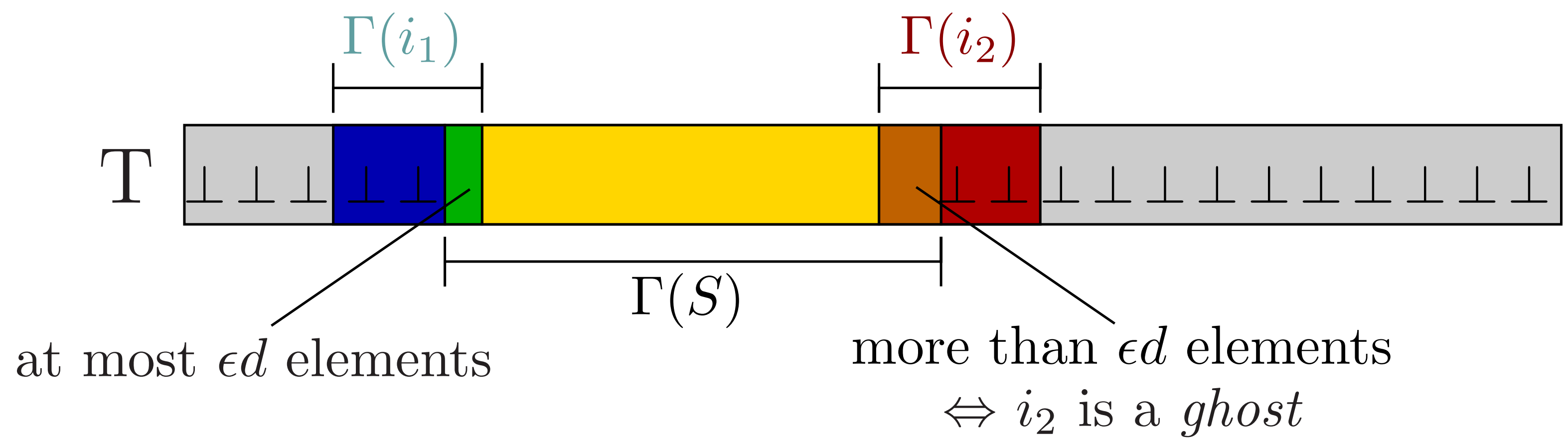
Let $S \subseteq \{1, \ldots, s\}$, $|S| = n$, be the subset of nonempty array entries.

For each index $i \in \{1, \ldots, s\}$ our data structure must satisfy:

- When $i \in S$:
  - $T[v]$.index $= i$ for a fraction $1 - \epsilon$ of $v \in \Gamma(i)$.
  - $T[v]$.index $\notin \{\neg i, \perp\}$ for $v \in \Gamma(i)$.
- When $i \notin S$:
  - $T[v]$.index $\in \{\neg i, \perp\}$ for a fraction $1 - \epsilon$ of $v \in \Gamma(i)$.
  - $T[v]$.index $\neq i$ for $v \in \Gamma(i)$.

Ghosts

Indices outside of S are easy to handle, except for a small set G of indices called the *ghosts* for S.



Lemma (Buhrman et al.):

In a $(2n + 1, d, \epsilon/2)$-expander there are at most n ghosts for S.

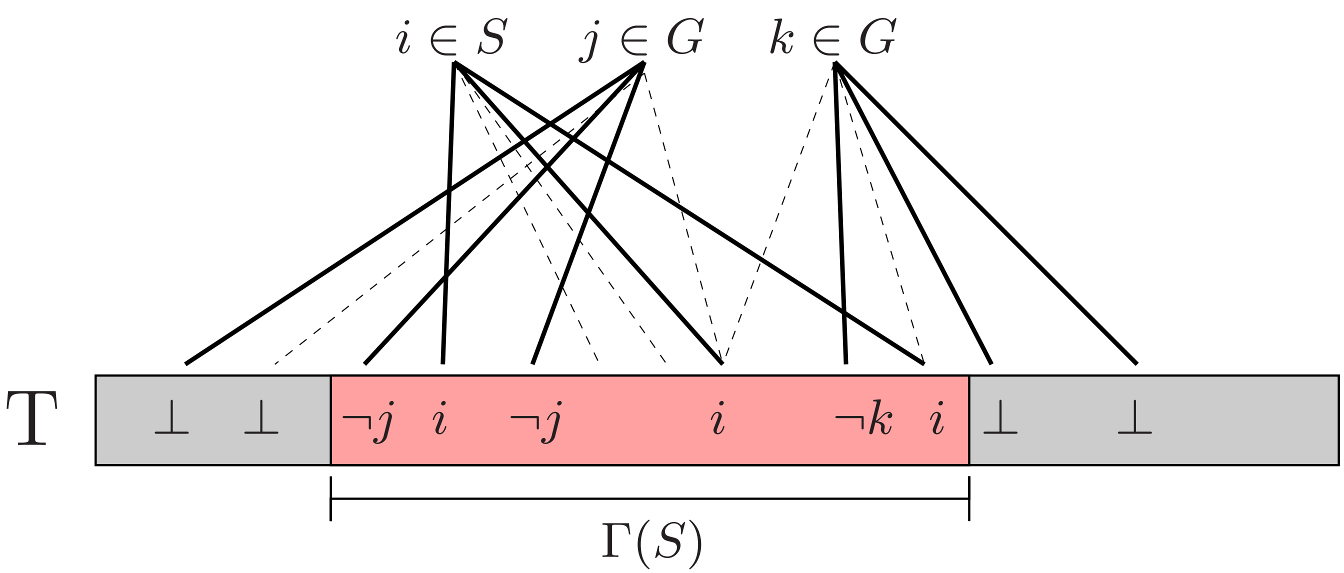
Assignments

Lemma:

In a $(2n, d, \epsilon)$-expander the indices in $S \cup G$ can be assigned $(1-\epsilon)d$ unique neighbors each.

Data structure initialization:

- For $i \in S$ put i in assigned neighbors.
- For $i \in G$ put $\neg i$ in assigned neighbors in $\Gamma(S)$.



Dynamization

Greedily maintaining the assignment:

- Whenever an index in $S \cup G$ has too few neighbors assigned to it, *all* its neighbors are assigned to it.
- Deterministic!

Analysis (similar to [Brodal-Fagerberg '99]):

- The greedy algorithm is $O(1)$-competitive with the optimal off-line number of reassignments.
- The assignment can be maintained off-line with $O(d)$ reassignments per insertion in S, amortized.

Busting ghosts

Big problem:

How are ghosts identified?

Solution:

- We care about ghosts only when they are looked up (since the expected lookup time is too large).
- More than $\log_{1/\epsilon} d$ random probes needed to find $\perp$ in $T[\Gamma(i)]$
  $\Rightarrow$ i is a ghost with probability $1 - O(1/d)$
  $\Rightarrow$ We check $T[\Gamma(i)]$ to see if i is a ghost.

Overview of results

Static data structure:

For any constant $\epsilon > 0$ there is a nonexplicit one-probe sparse array with success probability $1 - \epsilon$, using space $O(n \log s)$ words. (An explicit construction is possible in space $n \cdot 2^{O((\log \log s)^3)}$ words.)

Dynamization:

In the expander-augmented RAM model, we can perform a sequence of U updates and L lookups in the one-probe sparse array in time $O(U(\log s)^{1+o(1)} + L + t)$ with probability $1 - 2^{-\Omega(t/(\log s)^{1+o(1)})}$.

Lower bounds and open problems

Space lower bound:

Yao's $\Omega(\log n)$ lower bound on lookup time in implicit data structures implies that any equality-based approach must use space that grows with s.

Some open problems:

- Is the space usage optimal for one-probe schemes?
- Linear space usage by allowing a succinct hash function?
- Can Yao's lower bound be broken using space, say, $O(n \log^* s)$?