David R. Clark, Ian J. Munro.
Efficient Suffix Trees on Secondary Storage.
ACM-SIAM SODA 1996.

3.1 Partitioned Compact Pat Trees To control the accesses to secondary storage during searching we use the method suggested by Gonnet et al.[7]: decomposing the tree into disk block sized pieces (each called a partition). Each partition of the tree is stored using the CPT structure from the previous sections. The only change required to the CPT structure for storing the partitions is that the offset pointers in a block may now point to either a suffix in the text or a subtree (partition) so an extra bit is required to distinguish these two cases. We use a greedy bottom up partitioning algorithm and show that such a partitioning minimizes the maximum number of blocks accessed when traveling from the root to any leaf. The algorithm for building the index on secondary storage will be described in the full version of the paper.

The partitioning algorithm starts by assigning each leaf its own partition and a page depth of 1. Working upward, we apply the rules below at each node. A simple induction proof shows that the rules above produce a min-max optimal partitioning of the tree such that no other optimal partitioning has a smaller root block. The proof of optimality parallels the partitioning rules.

```
   If both children have the same page depth
(1)  if both children's partitions and the
            current node will fit in a block,
        merge the partitions of the children
            and add the current node
        set the page depth of the current
            node to that of the children
(2)  else
        close off the partitions of the
            children
        create a new partition for the
            current node
        set the page depth of the current
            node to one more than that of
            the children
  else
     close off the partition of the child
            with the lesser depth
(3)  if the current node plus the partition
            of the larger child will fit on
            a block
        add the current node to the child's
            partition
        set the page depth of the current
            node to match the child
(4)  else
        close off the partition of the
            remaining child
        create a new partition for the
            current node
        set the page depth of the current
            node to one more than that of
            the child
```

While the partitioning rules minimize the maximum number of secondary storage accesses, they can produce many small pages and poor fill ratios. There are several possible methods to alleviate this problem, including:

1. when a page is closed off, scan its children from smallest to largest to determine if they can be merged with the parent,
2. modify the rules to ensure a certain minimum fill ratio (e.g. all pages have to be 1/4 or 1/5 full),
3. pack multiple logical pages in each physical page,
4. ignore physical page boundaries when placing logical pages on disk.

Change one should be a part of any implementation of these rules. Change two will result in non-optimal partitioning in some cases but should be worthwhile in a practical system. The third technique should drastically minimize the storage requirements in practice but has a low guaranteed storage utilization and introduces some complications in the management of secondary storage. The last technique minimizes the storage requirements at a small cost for the potential transfer of an extra page of data on each access. In our current implementation for static text we use the first and fourth techniques.

We can bound the maximum number of pages traversed on any root leaf path in terms of the number of nodes in the Pat tree and the depth of the Pat tree.

THEOREM 3.1. *The page depth is less than* $1 + \left\lceil \frac{H}{\sqrt{p}} \right\rceil + \lceil 2 \log_p n \rceil$ *where* H *is the height of the Pat tree.*

Szpankowski shows that under very reasonable conditions on the text, H is logarithmic in n with probability one[22]. Linking these two results we obtain an expected performance bound logarithmic in n.

3.2 Empirical Results When producing the empirical results for indices on secondary storage, the optimal skip field sizes from the primary storage case were used for Holmes and the Bible (see Table 3). For the OED, a skip field size of six was used based on the experience with the Bible.

In each case, the depth of the tree is equal to the number of accesses to secondary storage needed to perform a search if the root block is held in memory. In the static case, we do not enforce page alignment so an access will consist of a seek followed by the transfer of at most two pages worth of data. The results above are for full suffix pointers. Truncating the suffix offsets would