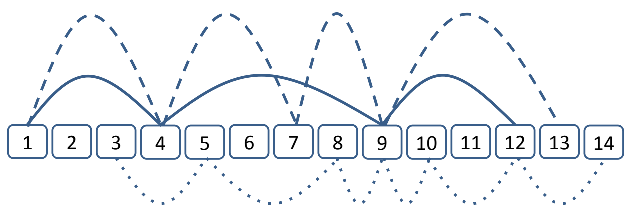

Figure 3: *Illustration of a linear order of data units and three example access requirements. Lines connect data blocks that belong to the same access requirement. The span of the access requirement shown in the solid line is 11.*

access requirement, the read head of the hard disk has to move from the first data block to the last irrespective of whether the intermediate blocks are read or skipped. Hence the span of an access requirement can be used as a measure of seek time - time taken to seek the last data unit starting from the first data unit. ~~We~~ use a relative probabilistic measure to include the frequency of use of each access requirement. Let I be the set of access requirements and A_i represent the span of the access requirement i. Let p_i be the probability that A_i will be used during rendering. We now define Estimated Seek Time (EST) as:

$$EST = \sum_{i \in I} p_i A_i \tag{1}$$

In this paper, we assume all access requirements are equally likely to be used thus all p_i values will be the same. We will use this to simplify the above equation to the following for our purposes.

$$EST = \sum_{i \in I} A_i.$$

It is important to note that the same measure can be used to describe the data transfer time. As mentioned earlier, whether the data between two required data units is read or skipped, the time taken to go from the first to the last required data unit is a measure of the delay caused by the disk. If all the intermediate data in the span is read, this time will be a measure of data transfer time, and if it is skipped, it is a measure of seek time~~. We~~ assume that only the required data is read and use this measure to quantify seek time.

The seek time is also measured in other works [Jiang et al. 2014; Jiang et al. 2013] as number of seeks and not parameterized using the distance between the required data units. In this work, we model seek time as the distance between the data units and optimize this measure. Using this measure, we show better performance than earlier works.

If we reduce the total EST in our optimization, then the average estimated seek time will be reduced. During optimization, we first choose and process the access requirement with the maximum span. ~~In the process~~, we not only reduce the average span, but also the maximum span, and hence the standard deviation in spans. This will in turn have an effect of providing consistent rendering performance with low data fetch delays as well as consistently small variation between such delays during rendering.

It is interesting to note that [Yoon et al. 2005] used span to measure the expected number of cache misses. Typically, with every cache miss, the missing data will be sought in the disk and fetched, thus adding to the seek time. In this aspect, using the span to measure the seek time is justified too.

3.3 Algorithm Overview

In [Yoon et al. 2005], the only allowed operation on the data units is the move operation and the optimal ~~solution~~ is computed using only that operation. For our purposes, we are allowed to copy data units, move them, and delete them if they are not used. Using these operations, our goal is to minimize EST while keeping the number of redundant copies as low as possible. After constructing a cache oblivious layout of the data set to get an initial ordering of data units, we copy one data unit to another location, and reassign one or more of the access requirements that use the old copy of the data unit to the new copy, such that the EST is reduced. If all the access requirements that used the old copy, now use the new copy of the data unit, then the old copy is deleted. We repeat this copying and possible deletion of individual data units until our redundancy limit has been reached.

Blocks to Copy: Note that the span of an access requirement does not change by moving an interior data unit to another interior location. Cost can be reduced only by moving the data units that are at the either ends of the access requirement. This observation greatly reduces the search space of data units to consider for copying. Additionally, for the sake of simplicity of the algorithm, we operate on only one data unit at a time.

Location to Copy: Based on the above observation, given an access requirement, we can possibly move the beginning or the end data units of an access requirement to its interior. This will reduce its span, thus reducing the EST for the layout. However, if the new location of the data unit is in the span of other access requirements, such as location 11 in Fig. 3, it increases the span of each of those accesses (all those three access requirements in Fig. 3) by one unit. Let j be the new location for the start or end data unit of an access requirement i. Let ΔA_i denote the change in the span of the access requirement i by performing this copying operation. Let k_j denote the number of access requirements whose span overlaps at location j. The reduction in EST by performing this copying operation is given by

$$\Delta EST_C^P(i,j) = \Delta A_i - k_j,$$

where C denotes *copying* the data unit for access requirement i to the location j; We put P in the term, since it is a partial term, which will be modified later. We find the location j where the start or end data unit of the access requirement i needs to be copied using a simple linear search through the span of i as

$$argmax_j(\Delta EST_C^P(i,j)).$$

Assignment of Copies to Access Requirements: The above operation would result in two copies of the same data unit, say d_{old} and d_{new}. Clearly the new copy d_{new} in location j will be used by the access requirement i. But d_{old} could be accessed by multiple other access requirements. All other access requirements that accesses d_{old} can either continue to use d_{old} or use d_{new} depending on the overall effect on their span. Let S be the set of access requirements whose span does not increase by using d_{new} instead of d_{old}. Now the total benefit by copying the data unit d_{old} of the access requirement i to the new location j is

$$\Delta EST_C(i,j) = \Delta A_i - k_j + \sum_{s \in S} \Delta A_s. \tag{2}$$

Moving versus Copying: Let T be the set of access requirements whose span will increase by accessing d_{new} instead of d_{old}. Further, let k_{old} be the number of access requirements in whose span d_{old} is. If we force all the access requirements that uses d_{old} to use