The Hardness of Cache Conscious Data Placement

Erez Petrank and Dror Rawitz
Technion

The growing gap between the speed of memory access and cache access has made cache misses an influential factor in program efficiency. Much effort has been spent recently on reducing the number of cache misses during program execution. This effort includes wise rearranging of program code, cache-conscious data placement, and algorithmic modifications that improve the program cache behavior. In this work we investigate the complexity of finding the optimal placement of objects (or code) in the memory, in the sense that this placement reduces the cache misses to the minimum. We show that this problem is one of the toughest amongst the interesting algorithmic problems in computer science. In particular, suppose one is given a sequence of memory accesses and one has to place the data in the memory so as to minimize the number of cache misses for this sequence. We show that if P $\neq$ NP, then one cannot efficiently approximate the optimal solution even up to a very liberal approximation ratio. Thus, this problem joins the small family of extremely inapproximable optimization problems. Two famous members in this family are minimum graph coloring and maximum clique. In light of this harsh lower bound only mild approximation ratios can be obtained. We provide an algorithm that can map arbitrary access sequences within such a mild ratio.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*Memory management (garbage collection)*; *Optimization* ; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; *Sequencing and scheduling*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Approximation algorithm, cache conscious code rearrangement, cache conscious data placement, computational complexity, hardness of approximation, memory management.

1. INTRODUCTION

1.1 Background and Previous Work

Much effort has recently been put into improving the cache performance of programs, i.e., reduce the number of cache misses. One avenue for improving cache behavior is to rearrange the code in memory so that cache misses during code fetch reduce (see for example [Schmidt et al. 1998; Henis et al. 1999; Gloy and Smith 1999]), a second avenue is to place data in the memory wisely using the memory manager (see for example [Chilimbi and Larus 1998; Calder et al. 1998; Chilimbi et al. 1999; Chilimbi et al. 1999]), and finally, algorithms may be modified so that

An extended abstract containing some of these results was presented at the 29th Annual ACM Symposium on Principles of Programming Languages (PoPL'02) [Petrank and Rawitz 2002].
This research was supported by generous funding from the Bar-Nir Bergreen Software Technology Center of Excellence - the Software Technology Laboratory (STL), and by the Technion V.P.R. Fund - Steiner Research Fund.
Authors' address: Department of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. Emails: {erez,rawitz}@cs.technion.ac.il.

they access the memory in a way that reduces cache misses (see for example [Boehm 2000; McKinley et al. 1996; Rivera and Tseng 1998; Intel 2001a; 2001b]).

In this work we concentrate on the first two avenues: cache conscious data and cache conscious code placement. The study of cache-conscious objects or code placement in the memory faces two major obstacles. The first obstacle is in obtaining the input data for the placement algorithm. A wise data placement requires knowing the sequence of accesses to the relevant objects. However, before a program is executed, it is not clear what lines of codes will be executed and in which order, so it is difficult to suggest the right arrangement of the code in the memory. Similarly, before a program is executed, it is hard to tell what the sequence of reads and writes of the program will be. So how can the memory manager decide wisely where to put an object that is now created? The second obstacle is the algorithm itself. Suppose one got the exact series of accesses to the memory, be it accesses to the program code or accesses to obtain the data. Can one do a good placement job given full knowledge of the future?

Looking at previous work, both problems are solved via smart heuristics. In order to obtain the data, Calder et al. [1998] use profiling. The assumption is that programs tend to have similar behavior even with varying inputs. Chilimbi and Larus [1998] on the other hand, use a different heuristic assumption. They build on the tendency of program to behave consistently over time. Thus, they gather information on the access behavior of the program during one period of time and then use it to improve the next period, assuming the behavior remains the same. Algorithms for code rearranging usually use smart static analysis to determine the code flow.

In order to determine how to place data (or code) in the memory, given the information gathered so far, previous work uses heuristic algorithms. Several such works manage to improve program performance significantly. The question arises, is there an efficient algorithm that does better? Is there an efficient algorithm that does best?

1.2 This Work

1.2.1 *The Hardness of data placement.* In this work we show that unless P = NP there is no efficient optimal algorithm for data placement or code rearrangement that minimizes the number of cache misses. Furthermore, it is not even possible to get close. Actually, in the worst case, all efficient algorithms output a placement that is very far from the optimal solution. This result holds in a very liberal sense. It holds even if one has completely overcome the difficulty of obtaining the sequence of future accesses, and is just trying to output a mapping of objects to the memory that does well for this specific given sequence. Consider an efficient algorithm that gets as input a sequence of memory accesses and outputs a mapping f of the objects in the sequence to the memory. Let f^* be the optimal mapping that achieves the minimal number of cache misses for this sequence. Let *Misses*(f) be the number of cache misses when the mapping f is used. We show that unless P = NP, then for any $\varepsilon > 0$ there exists a sequence for which $\mathrm{Misses}(f)/\mathrm{Misses}(f^*) = \Omega(n^{1-\varepsilon})$, where n is the length of the sequence. Thus, in the worst case, the algorithm does extremely poorly compared to the optimal mapping.

Our result holds for direct-mapped caches and also for t-way set-associative caches for any $t > 1$. It holds for any cache that has more than two cache blocks (all realistic caches do). It holds for any reasonable cache replacement algorithm[1]. It holds even in the (special) case where all objects have the same length, i.e., the difficulty does not stem from variable sized objects. It holds also if the placement of the objects is not required to be consecutive. Namely, the objects can be put anywhere in the memory. Finally, we stress again that this result holds even if one has learned the future accesses completely, and knows exactly what the sequence of accesses is going to be.

A corollary of our theorem is that these problems remain intractable even if one only tries to estimate the number of cache misses in an optimal placement, without outputting an actual placement. In particular, for every $\varepsilon > 0$, if $P \neq NP$, there is no efficient algorithm that outputs an estimate that is within a factor of $O(n^{1/2-\varepsilon})$ from the optimal value.

Similar results on the hardness of approximation, i.e., hardness to within an $O(n^{1-\varepsilon})$ ratio, are known to only a handful of interesting problems in computer science. The most famous of these are Minimum Graph Coloring (or, Minimum Chromatic Number) and Maximum Clique.

1.2.2 *The Hardness of Compressing the Information.* Next, we move to the other obstacle in cache conscious data placement: gathering the information. Most previous algorithms do not make use of the full sequence since it is too expensive. Thus, data is first processed to keep only pairwise information. For example, Calder et al. [1998] keep information on how many cache misses would be caused if a pair of objects o_i and o_j are placed in memory locations that are mapped to the same cache line. They do not keep further information that allows telling what would happen if three objects o_i, o_j, o_k are placed in this manner. Chilimbi and Larus [1998] gather information on the temporal affinity for each pair of objects. No information is kept on triplets.

A question that arises is: how good can we do with pairwise information? That is, do we lose valuable information in the process of reducing the full information to pairwise information? We show that in the worst case the lost information is critical. Namely, after the sequence is processed to keep only pairwise information, even an algorithm that has no time limitation cannot come close to the optimal solution. Clearly, with the full information at hand, an exponential time algorithm can find the best placement. It simply tries all possible placements[2]. But when we keep only information on pairs of objects, an exponential algorithm (and also doubly exponential algorithm, or any unlimited algorithm) will not be able to find a solution that is within a factor of $k - 3$ from the optimal solution (at least for some instances), where k is the number of blocks in the cache.

We now ask further: suppose that due to the cost of using the whole sequence one must rely on pairwise information. Can one efficiently find a solution that is good in the pairwise information setting? Namely, we are now looking for a solution that

[1]Note that if the replacement policy is extremely bad, then even an optimal mapping may become terribly expensive. In that case, outputting a random placement may do as bad as the optimum.
[2]As explained in Section 4.4 the optimal placement does not exploit the infinite memory. Searching for a placement that places all objects in a memory of size polynomial in the number of objects is enough.

behaves well with respect to pairs, even if it behaves poorly with respect to the whole sequence. We show that even this limited quest is difficult. Specifically, We show that, unless P = NP, there is no efficient optimal algorithm that minimizes the number of cache misses in the pairwise information setting. Moreover, as for the original problem, we show that one cannot find an approximate placement whose value is within a factor of $O(n^{1-\varepsilon})$ from the optimal value.

1.2.3 *In light of the lower bound.* In light of the harsh lower bound one may still be interested in the question of what can be done for an arbitrary access sequence. We provide approximation algorithms that find a placement whose value is within $\frac{n}{c \log n}$ (or $\frac{n}{c}$) from the optimal value for any $c > 0$. This is true for direct-mapped caches, for t-way set-associative caches, and for the pairwise information case. Obviously, our algorithms output solutions that are far from good, but, due to our lower bounds, we know that one cannot do much better. Approximation algorithms with similar mild approximation ratios were given for problems such as minimum graph coloring and maximum clique (see [Boppana and Halldórsson 1992; Halldórsson 1993]).

1.3 Implications

Our hardness results bear two important practical implications. First, there is no hope to find one (efficient) algorithm that does well for all possible benchmarks (unless P = NP). Therefore, one should study how normal programs behave and do well with respect to them, either by using heuristics, or by solving some special case of the general problem that corresponds to typical programs. The second important implication is that it is not possible to evaluate the potential of reducing cache misses via heuristics. Sometimes, researchers are interested in the potential benefits of a method. The potential is used to decide whether a project is worth the effort and later it is used to evaluate how much of the potential benefit is actually obtained by the strategy chosen. In this case, there is no hope to get even close to evaluating the potential reduction in cache misses (unless P = NP), and thus we cannot tell how much of the possible benefit a particular strategy obtains. We can only compare the cache conscious algorithm to the normal algorithm and check how much of an improvement is obtained.

The implication of the second result (about keeping information for pairs) is that if one wishes to spend an exponential time to find the optimal arrangement in the memory, say, because one has extremely short sequences that are reused many times, then one should be careful about the way data is processed. Keeping information in a pairwise manner is not good in this case: even if one has infinite time resources, pairwise information is not enough to even get close to an optimal solution.

Finally, researchers in the theory community may find it interesting that such a strong result can be obtained without the use of Probabilistically Checkable Proofs [Arora and Safra 1998; Arora et al. 1998]. Thus, this result could have been proven in the 70's. As far as we know, there is only a handful of (interesting) problems for which such strong results exist.

1.4 Background on Hardness of Approximation

A lot of research in theoretical computer science has been devoted to showing that some problems are hard to approximate. However, until the beginning of the 90's most of the results were fairly weak. The breakthrough was provided in a series of papers [Feige et al. 1991; Arora and Safra 1998; Arora et al. 1998] (see [Arora and Lund 1997] for more details). They contained a new characterization of NP in terms of *probabilistically checkable proofs* (PCP), and with it several strong results on the hardness of approximating interesting optimization problems. For example, using this characterization Arora et al. [1998] showed that for some $\varepsilon > 0$, there is no $(1-\varepsilon)$-approximation algorithm for MAX-3SAT, unless P = NP. Many other interesting problems were shown to be hard to approximate to within a constant. Also, hardness of approximation to within a logarithmic factor was then shown for *set cover* [Feige 1998]. Finally, several problems were shown to be hard to within a polynomial factor. The two most interesting ones were the maximum clique problem [Bellare et al. 1998; Håstad 1996], and the minimum chromatic number problem [Bellare et al. 1998; Feige and Kilian 1998]. For more background on approximation algorithms and their hardness the reader is referred to [Ausiello et al. 1999; Hochbaum 1997].

1.5 Techniques

We provide several reductions. The reductions are all from the k-colorability problem, which is known to be NP-complete for any $k \geq 3$. The input to each of the reductions is a graph and the output is a sequence of memory accesses. Our typical reduction has one very strong property. Specifically, we show that if the input graph is k-colorable, then the objects in the output sequence can be mapped to memory so that the number of cache misses is extremely small. However, if the input graph is not k-colorable, then any mapping of the objects in the output sequence to the memory causes a huge number of cache misses. This is true even if the input graph can be colored with just $k+1$ colors. We conclude that if one can determine quite vaguely what the number of cache misses of a given sequence would be for an optimal mapping, then the same algorithm can be used to tell if a graph is k-colorable.

1.6 Related Work

Our hardness result holds for direct-mapped caches and also for t-way set-associative caches for any $t > 1$. However, it does not apply to *fully associative caches*, since in this case the placement in the memory does not make a difference. In fact, when all objects are of the same size, the off-line problem of minimizing cache misses in the case of fully associative caches is solvable in polynomial-time [Belady 1966].

In this work we consider the problem of finding the best static placement for a given sequence of accesses. Namely, we assume that during execution the objects are not moved. Kennedy and Kremer [1998] discuss the use of dynamic data placement. They partition programs into phases, identify possible data placements for each phase, and remapping costs between phases, and then consider the problem of finding the data-layout that minimizes the total cost. They show that the problem of finding the best data-layout is NP-hard, and present a framework that finds the best data-layout using state of the art 0-1 integer programming solvers. Ding and Kennedy [1999a] apply run-time transformations that improve memory hierarchy performance.

Ding and Kennedy [1999b] present a global data transformation that does not use pair-wise information. Instead, it first splits and then selectively re-groups all data arrays in a program. Due to this transformation the compiler can improve cache-block spatial reuse

The first to study the NP-hardness of optimizing data placement was Thabit [1982]. He discussed the problem of co-locating objects to minimize the number of cache misses. Such a discussion is interesting even in the case of a one-block cache (i.e., where $k = 1$). He represented an access sequence by a *proximity* graph in which vertices correspond to objects and each edge between two objects is given a weight equal to the number of times the two objects appear adjacent in the access sequence. He has shown that an optimal placement may be obtained by finding a partitioning of the proximity graph into non empty subgraphs of equal size such that the weight of edges between the subgraphs is minimized. Then, he has shown that the problem of finding the best partitioning of a given graph is NP-hard.

Later, Hollander [1995] discussed the *k-positioning* problem in which the memory is finite (of size greater than the number of objects), and any memory block can be assigned to any one of k cache blocks. This problem is equivalent to our problem in the case of direct-mapped caches. Hollander also showed that the decision version of the k-positioning problem is NP-complete for any $k \geq 2$ and that the k-positioning problem is MAX SNP-hard for $k = 2$. Thus, when the cache has two blocks the problem cannot be approximated within a factor of $1 + \varepsilon$ for some $\varepsilon > 0$, unless P = NP. Our results are much stronger. First, we deal with realistic caches that have more than two blocks. Moreover, even if one is interested in caches of two blocks Hollander's result indicates that it is hard to approximate the problem to within some small constant, it does not rule out approximation by a factor of, say, 1.1. This means that there may exist an algorithm that outputs data placement that has at most 10% more misses than the optimum. Such an algorithm might have served as an excellent approximation in practice. Our results rule out the possibility of outputting a placement that is within a factor of $O(n^{1-\varepsilon})$ from the optimum for any $\varepsilon > 0$. An approximation not within such a ratio (i.e., an $\omega(n^{1-\varepsilon})$-approximation) cannot be considered useful in practice.

1.7 Organization

In Section 2 we present basic concepts in caching systems. In Section 3 we define the problems considered in the paper. In Section 4 we present and prove our main results on the hardness of approximating the problem of placing objects in memory such as to minimize cache misses. First, we discuss direct-mapped caches, and then we deal with t-way set-associative caches, for $t \geq 2$. In Section 5 we define pairwise information algorithms, and prove our results on using pairwise information: first, we show that pairwise information is may be misleading; then, we show that finding an approximate placement in the pairwise information setting is as hard as finding one for the original problem. Section 6 contains our approximation algorithms. In Section 7 we extend our results to a more general (and practical) setting. We conclude in Section 8.

2. CACHING SYSTEMS

Today's caching systems are extremely complex using coherence protocols, buffering, policies of dealing with read and write misses, etc. We will show that data placement is difficult even for the simplest system. A cache system includes a cache with k cache blocks, a.k.a. cache lines. The memory is normally much bigger than the cache, so it consists of many more blocks. When an access to the memory is made, there is a mapping function that determines where in the cache the memory block may be. If the block is found (valid) in the cache we have a *cache hit*. If it is not, there is a *cache miss* and the block is read from the memory into the cache block. The time it takes to bring the block from the memory to the cache is much longer than the time it takes to read a cache line into the CPU. Thus, the number of cache misses is considered an important factor in the running time of a program.

Most systems employ a direct-mapped cache, a two-way set-associative cache, or a four-way set-associative cache. In a *direct-mapped cache*, each memory block is mapped exactly to a single cache block. Of course, many memory blocks are mapped to any single cache block. Usually, the mapping is just the memory block number modulo the cache size, k. In a *t-way set-associative cache*, we have $k \cdot t$ cache blocks. The blocks are grouped into k sets of t blocks each. A memory block is mapped first to a cache set (i.e., its number modulo k determines the set), and then to one of t cache lines of this set. When an access to a memory block is made, the system calculates the set that corresponds to that memory block, and then either the memory line is available from one of the t cache lines of the set, or there is a cache miss and the block is read from the memory to one of the t corresponding cache blocks. A replacement protocol is used to determine which object is to be removed from the cache set when a cache miss occurs.

In today's technology cache blocks normally consists of 32, 64 or 128 bytes. Cache sizes start at 8KB for L1 caches and may reach several Mega Bytes for L2 caches in server machines.

For a more detailed explanation about memory hierarchy see [Hennessy and Patterson 1996].

3. THE PROBLEMS

In this section we define the problems we consider in the paper. We start with the simplest: cache conscious placement for direct-mapped cache systems.

Direct-Mapped Caches

Let us now try to formalize the problem in a theoretical framework. We denote the set of memory objects by $\mathcal{O} = \{o_1, \ldots, o_m\}$. We assume all objects have identical size. We denote a sequence of object accesses by $\sigma = (\sigma_1, \ldots, \sigma_n)$ where $\sigma_i \in \mathcal{O}$ for $i \in \{1, \ldots, n\}$. We assume the size of the memory is not bounded, i.e., infinite, and we start by assuming direct mapping to a cache of k lines (a.k.a. blocks). An address N in the memory is assigned by the direct mapping to block number $(N \bmod k)$ in the cache[3]. Given an input $(\mathcal{O}, \sigma)$ the goal is to find a memory placement of the objects in $\mathcal{O}$ that minimizes the number of cache misses which

[3]Our result holds for any other reasonable direct mapping. We require that the mapping be efficiently computable, efficiently invertible, and that memory blocks are evenly mapped into the

occur when accessing the data objects according to the sequence σ. Note that the placement of the objects in the memory completely determines the number of cache misses for the sequence σ. Once the objects have been placed, the direct mapping allows no freedom of choice during execution. Suppose that the objects in $\mathcal{O}$ are placed in the memory by the mapping f. Denote by $\text{Misses}((\mathcal{O},\sigma), f)$ the number of cache misses occurring while accessing the memory according to a sequence σ of objects from $\mathcal{O}$. We formalize the above in the following manner:

Minimum k-cache misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

Measure: $\text{Misses}((\mathcal{O},\sigma), f)$.

A simplification we make, is that we think of object sizes as being equal to each other and also equal to the size of the cache block. Note that this is a special case of the more general problem, and even this special case turns out hard. Our results can easily be extended to show hardness also for cases in which the objects are forced to being smaller or larger than a cache block. Another simplification we use is that the memory is not limited. Intuitively, the fact that actual memories are finite does not help in finding a good placement. We show this formally in Section 4.4. The third assumption we make is that objects are placed such that they are aligned. That is, the objects are placed in the beginning of memory blocks, and, therefore, they are mapped to the beginning of cache lines. Under this assumption, many feasible solutions are discarded. Thus, an optimal solution may be better than an aligned optimal solution. In Section 7 we give a formal definition of the unrestricted alignments version of the minimum k-cache misses problem, and we show that our results hold for the unrestricted alignments case as well.

t-way Set-Associative Caches and Sensible Replacement Protocols

A natural extension of the k-cache misses problem is the *t-way k-cache misses* problem. In this problem we are given a t-way set-associative cache: each of the k cache sets contains t cache blocks. This means that, at any given moment, there can be t objects in each cache set (when using our simplified object sizes). Here, the number of misses is determined by the mapping of objects to the memory and by the policy, called *the replacement protocol*, used to determine which of the t blocks in the set is replaced when a miss occurs.

Our results for t-way set-associative caches require that the replacement policy is not extremely bad. If it is, we may get that all mapping of objects to memory behave so bad, so that there is no difference between an optimal mapping and any other mapping. We think of a replacement policy as extremely bad, if recently used objects keep being replaced, whereas unused objects are kept in the cache.

Consider the following scenario. We are given a sequence of memory accesses ending with many repetitions of a sub-sequence $\tilde{\sigma}$ that accesses a subset A of

k cache lines. In fact, an even distribution is not necessary, but being drastically far from even distribution may be a problem. Of course, this never happens in real systems.

the objects, and all objects in A are mapped to the same cache set. Clearly, if the number of objects in A is greater than t many cache misses will occur. However, what should we expect when there are t or fewer objects in A? A bad replacement protocol may keep old memory objects in the cache and replace the objects in A even though they repeat many times in this sub-sequence. This sort of behavior is very expensive in terms of cache misses and does not happen in practical replacement protocols. We expect a reasonable replacement protocol to place all the objects in A in the cache and prevent more cache misses. Therefore, a reasonable requirement from a replacement protocol of a t-way set-associative cache is to satisfy the following rule: if a (possibly very long) sequence iteratively accesses q objects which are mapped to the same cache set, and $q \leq t$, then the number of cache misses should not exceed q by much. We formalize this by the following.

Definition 3.1. A replacement protocol for a t-way set-associative cache is called *C-sensible* if the following holds. Let σ be a sequence of q distinct objects $\sigma = (o_{i_1}, o_{i_2}, \ldots, o_{i_q})$ and let these objects be placed in the memory via any placement that does not cause more than t of these objects to be mapped to the same cache set. Then, for any possible sequence σ' and any integer ℓ, when we execute the sequences σ' and σ^ℓ consecutively (i.e., σ' followed by ℓ iterations of σ) we get at most $q + C$ misses on the accesses of σ^ℓ.

In asymptotic analysis, the exact value of the constant C becomes immaterial. Therefore, we will sometimes (when appropriate) ignore the constant C and call the replacement protocol *sensible*.

Clearly, replacement protocols such as LRU and FIFO are 0-sensible replacement protocols. A pseudo-LRU algorithm is used in the Intel Pentium platform. The replacement protocol works as follows. Each set is divided into two pairs. Each pair has a flag indicating which block in this pair was accessed last. Another flag is used to determine which pair contains the most recently used block. The replacement protocol replaces the least recently used block from the least recently used pair (note that this may not be the least recently used block). This replacement protocol is 0-sensible as well.

t-way Set-Associative Caches

Denote by $\text{Misses}_t((\mathcal{O}, \sigma), f)$ the number of cache misses occurring while accessing the memory according to a sequence σ of objects from $\mathcal{O}$ when using a t-way set-associative cache (the replacement protocol should be clear from the context). The cache misses minimization problem in this case can be formulated as follows:

Minimum t-way k-cache misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

Measure: $\text{Misses}_t((\mathcal{O}, \sigma), f)$.

We define the following for the t-way k-cache misses problem. The corresponding definitions for the k-cache misses problem (and for other problems that will be defined later) can be understood in a straightforward manner. Given a set of objects $\mathcal{O}$ and a sequence σ, a placement f^* is called *optimal* if every placement f satisfies $\text{Misses}_t((\mathcal{O},\sigma),f^*) \leq \text{Misses}_t((\mathcal{O},\sigma),f)$. We sometimes refer to the value of an optimal solution with respect to an instance $I = (\mathcal{O},\sigma)$ as $\text{Opt}(I)$. A placement f is called *r-approximate* if $\text{Misses}_t((\mathcal{O},\sigma),f) \leq r \cdot \text{Opt}(\mathcal{O},\sigma)$. A polynomial time algorithm is called an *r-approximation algorithm* if, given an instance $(\mathcal{O},\sigma)$, it returns an r-approximate placement.

Graph Colorability

We use the graph k-colorability problem in our reductions.

Graph k-colorability (kCOL)

Instance: An undirected graph $G = (V,E)$.

Problem: Is G k-colorable, i.e., does there exist a function $c : V \to \{0,\ldots,k-1\}$ such that $c(u) \neq c(v)$ whenever $(u,v) \in E$?

This problem is known to be NP-complete for any $k \geq 3$ (see, e.g., [Garey and Johnson 1979]).

A function $c : V \to \{0,\ldots,k-1\}$ is called a *k-coloring*. Given a graph G and a k-coloring c we call an edge $e = (u,v)$ monochromatic if $c(u) = c(v)$. We denote by $\text{Mono}(G,c)$ the number of monochromatic edges in G. In particular, if G is k-colorable, then there exists a k-coloring of G, c, such that $\text{Mono}(G,c) = 0$.

4. HARDNESS RESULTS

In this section we prove our results on the difficulty of approximating minimum k-cache misses and minimum t-way k-cache misses. In both problems we show that, if P $\neq$ NP, for every $k \geq 3$ the problem cannot be approximated within $O(n^{1-\varepsilon})$ for every $\varepsilon > 0$. We show this by constructing reductions from graph k-colorability. The reductions get as input a graph G and output an instance of the k-cache misses problem or the t-way k-cache misses problem such that if G is k-colorable then the optimal number of cache misses is extremely low, and otherwise it is very high. Such reductions imply that an approximation algorithm for the k-cache misses problem or the t-way k-cache misses problem can be used to distinguish between graphs which are k-colorable and graphs which are not. Moreover, even a vague evaluation of the optimal solution provides a way to determine whether a given graph is k-colorable.

4.1 Notation and Terminology

We use the following notation. Given two sequences $\sigma = (\sigma_1,\ldots,\sigma_n)$ and $\sigma' = (\sigma'_1,\ldots,\sigma'_{n'})$, we denote by $\sigma \bigodot \sigma'$ the sequence $(\sigma_1,\ldots,\sigma_n,\sigma'_1,\ldots,\sigma'_{n'})$ (i.e., $\bigodot$ denotes concatenation). σ^t stands for $\bigodot_{i=1}^{t} \sigma$.

4.2 Minimum k-cache Misses

We start by constructing an instance of the k-cache problem for any possible graph. This construction will later be used in the proof of Theorem 4.4. In the construction we use a parameter ℓ which is a constant to be determined later. Given a graph

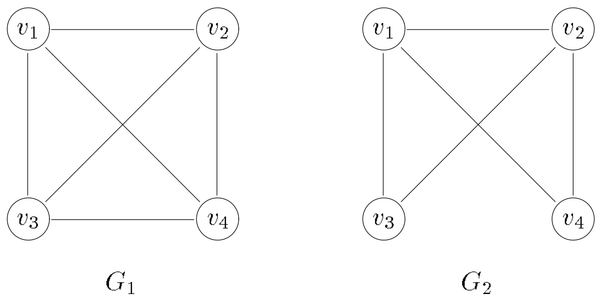

Fig. 1. Example: G_2 is 3-colorable, while G_1 is not.

$G = (V, E)$ we construct an instance of the k-cache misses problem. This instance contains a memory object for each vertex in V, and a sequence of accesses in which each edge is represented by a sub-sequence. The sub-sequence corresponding to an edge $(u, v) \in E$ consists of many repetitions of the two objects which correspond to the two vertices u and v. Formally, given a graph $G = (V, E)$ we define $H_\ell(G) = (\mathcal{O}_G, \sigma_G)$, where $\mathcal{O}_G = \{o_1, \ldots, o_{|V|}\}$, and

$$\sigma_G = \bigodot_{(v_i, v_j) \in E} (o_i, o_j)^{|E|^\ell} .$$

We assume there exist some canonical ordering of the edges, otherwise σ_G is defined in a way which is not unique. In fact, any ordering suffices for the correctness of our results.

Since ℓ is a constant $H_\ell(G)$ can be constructed in polynomial time, and it is of size $\Theta(|E|^{\ell+1})$.

Example 4.1. Consider the instances for the 3-colorability problem that are given in Figure 1. Clearly, G_2 is 3-colorable, while G_1 is not. Let $H_\ell(G_1) = (\mathcal{O}_1, \sigma_1)$, and $H_\ell(G_2) = (\mathcal{O}_2, \sigma_2)$ be the corresponding instances for the 3-cache misses problem. Both $\mathcal{O}_1$ and $\mathcal{O}_2$ contain four objects o_1, o_2, o_3 and o_4. However, σ_1 and σ_2 are different:

$$\begin{aligned}
\sigma_1 &= (o_1, o_2)^{6^\ell} \bigodot (o_1, o_3)^{6^\ell} \bigodot (o_1, o_4)^{6^\ell} \bigodot (o_2, o_3)^{6^\ell} \bigodot (o_2, o_4)^{6^\ell} \bigodot (o_3, o_4)^{6^\ell} \\
\sigma_2 &= (o_1, o_2)^{5^\ell} \bigodot (o_1, o_3)^{5^\ell} \bigodot (o_1, o_4)^{5^\ell} \bigodot (o_2, o_3)^{5^\ell} \bigodot (o_2, o_4)^{5^\ell}
\end{aligned}$$

Given four objects and only three cache lines, we have to assign at least two objects to memory addresses that correspond to the same cache line. Thus, the number of misses due to σ_1 is at least 6^ℓ. On the other hand, if our placement maps o_3 and o_4 to the same cache line, while mapping o_1 and o_2 to the remaining two cache lines (one each), the number of misses due to σ_2 will be 6.

Next we relate placements to k-colorings in such a way that the number of misses caused by the sequence σ_G when using a placement f will be strongly correlated to the number of monochromatic edges in G when colored by a corresponding k-coloring c. Given a k-coloring c of V we define a family of corresponding placements F_c. A placement $f : \mathcal{O}_G \to \mathbb{N}$ is in F_c if it places o_i in a memory block that is mapped to a cache line which is indexed by $c(v_i)$ (e.g., $f(o_i) = i \cdot k + c(v_i)$). Note that for every placement f there exists a k-coloring c for which $f \in F_c$ (for c that colors $v_i \in V$ according to the cache line assigned to the memory address in which o_i is placed). The next lemma shows a connection between a k-coloring c of V and a corresponding placement $f \in F_c$. Recall that $\mathrm{Mono}(G, c)$ is the number of monochromatic edges in G when colored by c.

LEMMA 4.2. *For any k-coloring c of V, and for any $f \in F_c$,*

$$\mathit{Misses}(H_\ell(G), f) = \Theta(|E|^\ell \, \mathit{Mono}(G, c)) + O(|E|) .$$

PROOF. Consider a mapping $f \in F_c$. Now, examine an edge $(v_i, v_j) \in E$. If the edge (v_i, v_j) is monochromatic then $f(o_i)$ and $f(o_j)$ are mapped to the same cache line. Therefore, the sub-sequence $(o_i, o_j)^{|E|^\ell}$ that corresponds to a monochromatic edge contributes at least $2|E|^\ell - 1$ misses. On the other hand, if (v_i, v_j) is not monochromatic then $f(o_i)$ and $f(o_j)$ are not mapped to the same cache line. Thus, a non-monochromatic edge causes no more than two misses. $\square$

Next, we show that if graph G is k-colorable then the optimal number of cache misses is low, and otherwise it is very high.

PROPOSITION 4.3. *If a graph G is k-colorable then $\mathit{Opt}(H_\ell(G)) = O(|E|)$, otherwise $\mathit{Opt}(H_\ell(G)) = \Omega(|E|^\ell)$.*

PROOF. If G is k-colorable then there exists a k-coloring c for which there are no monochromatic edges. Thus, by Lemma 4.2, $\mathrm{Misses}(H_\ell(G), f) = O(|E|)$ for $f \in F_c$. On the other hand, if G is not k-colorable then a k-coloring for which there are no monochromatic edges does not exist, which means that $\mathrm{Mono}(G, c) > 0$ for any k-coloring c of G. As mentioned before, for any placement f there exists a k-coloring c such that $f \in F_c$. Therefore, Lemma 4.2 implies that $\mathrm{Misses}(H_\ell(G), f) = \Omega(|E|^\ell)$ for every placement f. $\square$

THEOREM 4.4. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the k-cache misses problem cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

PROOF. Given a graph G we fix $\ell = \lceil \frac{3}{\varepsilon} - 1 \rceil$ (note that it is a constant as promised), and construct $H_\ell(G)$. By Proposition 4.3 if $G \in k\mathrm{COL}$ then $\mathrm{Opt}(H_\ell(G)) = O(|E|)$, while if $(G, k) \notin \mathrm{COL}$ then $\mathrm{Opt}(H_\ell(G)) = \Omega(|E|^\ell)$. Thus, distinguishing between instances I for which $\mathrm{Opt}(I) = O(|E|)$, and instances I for which $\mathrm{Opt}(I) = \Omega(|E|^\ell)$ is NP-hard. Suppose there exists an algorithm that outputs an $O(|E|^{\ell-2})$-approximate placement. Then, we can use it to determine if $\mathrm{Opt}(I) = O(|E|)$. If there exists a placement f such that $\mathrm{Misses}(H_\ell(G), f) = O(|E|)$ then the algorithm will output a placement g such that $\mathrm{Misses}(H_\ell(G), g) = O(|E|^{\ell-1})$. However, if such a placement does not exist, the algorithm will output a placement g for which $\mathrm{Misses}(H_\ell(G), g) = \Omega(|E|^\ell)$. This implies that there cannot exist an $O(|E|^{\ell-2})$-approximation algorithm for the minimum cache misses problem, unless $\mathrm{P} \neq \mathrm{NP}$.

It remains to relate $|E|^{\ell-2}$ to the instance size, i.e., to $|H_\ell(G)|$. Note that:

$$\begin{aligned}|E|^{\ell-2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{\ell+1}})\\ &= \Theta(|H_\ell(G)|^{1-\frac{3}{\ell+1}})\\ &= \Theta(|H_\ell(G)|^{1-\frac{3}{\lceil 3/\varepsilon\rceil}})\\ &= \Omega(|H_\ell(G)|^{1-\varepsilon})\end{aligned}$$

and we are done. □

Corollary 4.5. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the k-cache misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof. Given a graph G we fix $\ell = \lceil \frac{3}{2\varepsilon} - 1 \rceil$, and construct $H_\ell(G)$. The gap shown in the proof of Theorem 4.4 implies that if we can efficiently approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$ then we can decide in polynomial time whether G is k-colorable or not. Thus, it is NP-hard to approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$, and

$$\begin{aligned}|E|^{(\ell-2)/2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{2(\ell+1)}})\\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\frac{3}{2(\ell+1)}})\\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\frac{3}{2\lceil 3/2\varepsilon\rceil}})\\ &= \Omega(|H_\ell(G)|^{\frac{1}{2}-\varepsilon})\ .\end{aligned}$$

The corollary follows. □

4.3 Minimum t-way k-cache Misses

We now turn to showing that data placement is difficult even for systems that do not employ direct mapping. We start with constructing an instance of the t-way k-cache misses problem (i.e., a sequence of accesses) for any given graph $G = (V, E)$. As in the case of the k-cache misses problem this instance contains a memory object for each vertex in V, and each edge $(u, v) \in E$ is represented by some sub-sequence which consists of many repetitions of the two objects which correspond to the two vertices u and v. However, as opposed to the direct mapping case, the instance contains more (dummy) memory objects and an additional sub-sequence whose purpose is to force the replacement policy to imitate a direct-mapped cache. Due to this property we can establish a connection between the number of monochromatic edges in a given k-coloring and the number of cache misses in a corresponding mapping for any sensible replacement protocol.

Given a graph $G = (V, E)$, we construct the instance $H_\ell(G) = (\mathcal{O}_G, \sigma_G)$, such that

$$\mathcal{O}_G = \{o_1, \ldots, o_{|V|}\} \cup \{d_{i,j} \ : i \in [k],\ j \in [t+1]\}$$

where $[r] \triangleq \{0, \ldots, r-1\}$, and

$$\sigma_G = \sigma_{G,1} \bigodot \sigma_{G,2}$$

where

$$\sigma_{G,1} = \bigodot_{(v_i,v_j)\in E} \left(\left(\bigodot_{i'=0}^{k-1} \bigodot_{j'=0}^{t-2} (d_{i',j'}) \right) \bigodot (o_i, o_j) \right)^{|E|^\ell}$$

and

$$\sigma_{G,2} = \bigodot_{j=0}^{t} \left(\bigodot_{i=0}^{k-1} \bigodot_{j'\neq j} (d_{i,j'}) \right)^{|E|^{\ell+2}}$$

Again, we assume there exists some canonical ordering of the edges, otherwise $\sigma_{G,1}$ is not uniquely defined. Any ordering suffices for the correctness of our results.

Next, as in the k-cache case we relate mappings to k-colorings in such a way that the number of misses in σ_G due to a mapping will be strongly related to the number of monochromatic edges in G when colored by the corresponding k-coloring. Given a k-coloring c of V we define a corresponding family of placements F_c. A placement $f : \mathcal{O}_G \to \mathbb{N}$ is in F_c if it places o_i in a memory block that is mapped to a cache set which is indexed by $c(v_i)$, and distributes the objects in $D_j \triangleq \{d_{i,j} : i \in [k]\}$ into the k different cache sets, i.e., places, for all $j \in [t+1]$, exactly one object from the set D_j in each cache set. Note that for every placement f which places exactly one object from D_j in each cache set for all $j \in [t+1]$ there exists a k-coloring c for which $f \in F_c$ (the one which colors $v_i \in V$ according to the cache set assigned to the memory address in which o_i is placed). The next lemma shows a connection between a k-coloring c of V and a corresponding placement $f \in F_c$.

Lemma 4.6. *For any k-coloring c of V, if the replacement protocol is C-sensible, then for any $f \in F_c$,*

$$Misses_t(H_\ell(G), f) = \Theta(|E|^\ell \, Mono(G,c)) + O(|E|) \ .$$

Proof. Consider a mapping $f \in F_c$. When using a cache with an C-sensible replacement protocol and the placement f the number of misses in the second part of σ_G (i.e., in $\sigma_{G,2}$) is at most $t \cdot C = O(1)$.

Now let us check $\sigma_{G,1}$. Consider an edge $(v_i, v_j) \in E$. If (v_i, v_j) is monochromatic then $f(o_i)$ and $f(o_j)$ are mapped to the same cache set. In this case, each repetition of the sub-sequence $\tilde{\sigma} = (\bigodot_{i'=0}^{k-1} \bigodot_{j'=0}^{t-2} (d_{i',j'})) \bigodot (o_i, o_j)$ causes at least one cache miss (and up to $t+1$ misses) because there are $t+1$ objects in this sub-sequence which are mapped to the same cache set. Since $\tilde{\sigma}$ is repeated $|E|^\ell$ times there are $\Theta(|E|^\ell)$ misses due to the sub-sequence corresponding to the edge (v_i, v_j). On the other hand, if the edge (v_i, v_j) is not monochromatic then $f(o_i)$ and $f(o_j)$ are not mapped into the same cache set. In this case, there are exactly t objects whose addresses are mapped to the i'th set for any $i \in [k]$. Thus, when using a sensible replacement protocol, the sub-sequence $\tilde{\sigma}^{|E|^\ell}$, which corresponds to the non-monochromatic edge (v_i, v_j), causes $O(1)$ misses.

The lemma follows from the fact that there are $\mathrm{Mono}(G, c)$ monochromatic edges, and up to $|E|$ non-monochromatic edges. □

Proposition 4.7. *If the replacement protocol is sensible, then if G is k-colorable then $Opt(H_\ell(G)) = O(|E|)$, otherwise $Opt(H_\ell(G)) = \Omega(|E|^\ell)$.*

PROOF. Consider an optimal placement f. There are two possibilities, either, for all $j \in [t+1]$, f splits each set $S_j \triangleq \{d_{i,j'} : i \in [k], j' \in [t+1]\} \setminus D_j$ evenly in the cache, i.e., for each cache set C_r ($r \in [k]$) f places exactly t objects from S_j at addresses corresponding to C_r, or f does not have this property. If f does not place the $d_{i,j}$'s as described above then for some j more than t objects from S_j are placed at addresses corresponding to some cache set. In this case, the resulting number of misses in $\sigma_{G,2}$ will be $\Omega(|E|^{\ell+2})$. On the other hand, if f places the $d_{i,j}$'s in this way the number of misses caused by $\sigma_{G,2}$ will be no greater than $O(tk) = O(1)$ (k and t are constants). By Lemma 4.6, the overall number of misses will be $O(|E|^{\ell+1}) = o(|E|^{\ell+2})$. We conclude that an optimal placement must have the above property, i.e., it must place exactly t objects from S_j at addresses corresponding to each cache set for every $j \in [t+1]$.

Now, we show that in order to satisfy the above requirement, an optimal placement must place exactly one object from $D_j = \{d_{i,j} : i \in [k]\}$ at an address corresponding to each cache set for any $j \in [t+1]$. Consider two indices $j_1, j_2 \in [t+1]$ such that $j_1 \neq j_2$. We know that an optimal placement f must place exactly t objects from S_{j_r} at addresses corresponding to each cache set for $r = 1, 2$. By definition $S_{j_2} = S_{j_1} \cup D_{j_1} \setminus D_{j_2}$. Thus, the surplus in any cache set after adding the objects in D_{j_1} to S_{j_1} must be eliminated by the removal of D_{j_2}. Thus, the number of objects from the sets D_{j_1} and D_{j_2} which are placed by f in each cache set must be the same. This argument works for all $j_1, j_2 \in [t+1]$ such that $j_1 \neq j_2$, and, therefore, the number of objects from the set D_j which are placed by an optimal placement f in each cache set must be the same for any $j \in [t+1]$. We claim that the only way to do this while placing exactly t objects from S_j in each cache set for every $j \in [t+1]$ is by placing exactly one object from D_j in each cache set for any $j \in [t+1]$. Assume the contrary. Then, there exist a cache set i in which f places either no objects from the D_j's or more than one object from each of the D_j's. Pick an index $j \in [t+1]$ and examine the set S_j. In the first case no objects are placed in the i'th set, and in the second at least $2t$ objects are placed in the i'th set. Both in contradiction to the requirement that an optimal solution places exactly t objects from S_j in each cache set for every $j \in [t+1]$. Therefore, an optimal solution must place exactly one object from D_j in each cache set for any $j \in [t+1]$.

The rest of the proof is the similar to the proof of Proposition 4.3 (where Lemma 4.6 replaces Lemma 4.2). Recall that if a placement f places exactly one object from D_j in each cache set for any $j \in [t+1]$, then there exist a k-coloring c such that $f \in F_c$. If G is k-colorable then there exists a k-coloring c for which there are no monochromatic edges. Thus, by Lemma 4.6, $\text{Misses}_t(H_\ell(G), f) = O(|E|)$ for any $f \in F_c$. On the other hand, if G is not k-colorable then a k-coloring for which there are no monochromatic edges does not exist. Therefore, Lemma 4.6 implies that $\text{Misses}(H_\ell(G), f) = \Omega(|E|^{\ell})$ an optimal placement f. $\square$

THEOREM 4.8. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the t-way k-cache misses problem (when using a sensible replacement protocol) cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless P = NP.*

PROOF. Given a graph G we fix $\ell = \lceil \frac{4}{\varepsilon} - 2 \rceil$, and construct $H_\ell(G)$. By Proposition 4.7 distinguishing between instances I for which $\text{Opt}(I) = O(|E|)$, and instances I for which $\mathrm{Opt}(I) = \Omega(|E|^{\ell})$ is NP-hard. This implies that there cannot exist an $O(|E|^{\ell-2})$-approximation algorithm for the minimum cache misses problem, unless P $\neq$ NP. It remains to relate $|E|^{\ell-2}$ to the instance size, i.e., to $|H_\ell|$. Note that

$$\begin{aligned} |E|^{\ell-2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{\ell+2}}) \\ &= \Theta(|H_\ell(G)|^{1-\frac{4}{\ell+2}}) \\ &= \Theta(|H_\ell(G)|^{1-\frac{4}{\lceil 4/\varepsilon \rceil}}) \\ &= \Omega(|H_\ell(G)|^{1-\varepsilon}) \end{aligned}$$

and we are done. $\square$

Corollary 4.9. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the t-way k-cache misses problem (when using a sensible replacement protocol) cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof. Given a graph G we fix $\ell = \lceil \frac{2}{\varepsilon} - 2 \rceil$, and construct $H_\ell(G)$. The gap shown in Theorem 4.8 implies that if we can efficiently approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$ then we can decide in polynomial time whether G is k-colorable or not. Thus, it is NP-hard to approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$, and

$$\begin{aligned} |E|^{(\ell-2)/2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{2(\ell+2)}}) \\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\frac{4}{2(\ell+2)}}) \\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\frac{2}{\lceil 2/\varepsilon \rceil}}) \\ &= \Omega(|H_\ell(G)|^{\frac{1}{2}-\varepsilon}) \ . \end{aligned}$$

The corollary follows. $\square$

Note that the proof of Lemma 4.6 and consequently Theorem 4.8 and Corollary 4.9 hold for any sensible replacement protocol, and in particular, for LRU, FIFO, and pseudo-LRU.

4.4 Finite Size Memory

It seems intuitively correct that an infinite memory makes the placement problem easier to solve, and our hardness results stronger. In this section we formalize this intuition by showing that our hardness results are valid even if the memory is finite. We start by integrating the memory size into the definition of the problem, and continue by showing that the modified problem is as hard to approximate as its infinite counterpart. For simplicity we apply this modification to the simplest model, i.e., a direct-mapped cache with the mapping being the modulus function. A similar argument holds also for the t-way set-associative case, and for any other mapping that maps the memory to the cache in a reasonably balanced manner.

Consider the following scenario. We are given a set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, an access sequence σ, and a memory of size $M \geq |\mathcal{O}|$. Our goal is to find a placement $f : \mathcal{O} \to [M]$ which minimizes the number of cache misses. The problem is formalized as follows.

Minimum k-cache M-memory misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$, and an integer $M \geq m$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to [M]$.

Measure: $\mathrm{Misses}((\mathcal{O}, \sigma), f)$.

Theorem 4.10. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the k-cache M-memory misses problem cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

Proof. We use the same proof as in Theorem 4.4, except that on a given input graph G, the output of the reduction is $(H_\ell(G), m \cdot k)$. namely, we add a bound on the memory size being $M = m \cdot k$, where m is the number of objects and k is the number of blocks in the cache. To see that this works well, we have to show that when G is k-colorable, the optimal placement has a small number of cache misses and when G is not k-colorable any placement causes many cache misses. In the latter case the analysis does not change. The limitation on the placement (i.e., the size of the memory) may only increase the cache misses for the optimal solution, and this does not foil the analysis. In the first case, where G is k-colorable, we need to show that the optimal placement does have the low number of cache misses as in the original proof of the theorem. However, since there are m objects, the optimal placement may try to map at most m objects to any specific cache block. Doing this is possible also when the memory has $m \cdot k$ cache blocks. To map j objects to cache block number i (for $j \leq m$), the placement puts them in memory blocks $i, i+k, \ldots, i+j \cdot k$. Thus, the optimal placement is possible also when the memory is restricted, and we are done with the proof of Theorem 4.10. □

Corollary 4.11. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the k-cache M-memory misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof. Same argument as in Corollary 4.5. □

5. PAIRWISE INFORMATION

In this section we show that keeping only pairwise information on a given sequence of memory accesses causes (in the worst case) a loss of critical information, even if one is allowed to process the information with no computational limitations. We also show that even if one is forces to use pairwise information, finding a good placement with respect to pairwise information (and not necessarily with respect to the whole sequence) is difficult.

Our intentions is to model the actual procedure by which data placement algorithms have kept the data [Chilimbi and Larus 1998; Calder et al. 1998]. Since the full sequence of accesses is hard to predict, and, even when given, it is long and requires a lengthy processing, previous works used pairwise information. In particular, for each pair of objects – how many cache misses would result from their being placed to two locations in the memory that are mapped to the same cache line.

We show that the value of a pairwise optimal solution may be far from the value of an optimal solution to the k-cache misses problem for every $k \geq 2$. We do this by constructing an instance with the following property: a placement that achieves the minimum cache misses for pairs is a factor of $k-3$ away from an optimal placement.

Then, we show that, unless P = NP, there is no efficient optimal algorithm that minimizes the number of cache misses in the pairwise information setting. Moreover, we show that one cannot find an approximate placement whose value is within a factor of $O(n^{1-\varepsilon})$ from the optimal value.

5.1 The Problem

Let us begin with formalizing the above notion of using pairwise information. Consider an instance $(\mathcal{O}, \sigma)$ of the k-cache misses problem, where $\mathcal{O} = \{o_1, \ldots, o_m\}$. Given two objects $o_i, o_j \in \mathcal{O}$, let $\sigma_{i,j}$ be the sequence resulting from eliminating all other elements, i.e., all elements o_r such that $r \neq i, j$ from σ. We call the sequence $\sigma_{i,j}$ the *pairwise sequence* of o_i and o_j, or the *projection* of σ on o_i and o_j. Consider the following scenario: we are given $\binom{m}{2}$ pairwise sequences $\{\sigma_{i,j} : 1 \leq i < j \leq m\}$, and we try to find one global good mapping that minimizes the number of misses in all pairwise sequences, that is, we try to find a mapping f that minimizes $\text{Pairs}((\mathcal{O}, \sigma), f) = \sum_{i<j} \text{Misses}((\{o_i, o_j\}, \sigma_{i,j}), f)$. Formally, we consider the following problem:

Minimum k-cache pairwise misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

Measure: $\text{Pairs}((\mathcal{O}, \sigma), f)$.

Note that instances of the k-cache misses problem and instances of the k-cache pairwise misses problem have the same format. The difference is in the target optimization function.

Given a placement f let $\text{Size}_{\mathcal{O},f}(i)$ be the number of objects from $\mathcal{O}$ which are mapped by f to the i'th cache line. Using the convention that memory blocks are mapped to cache blocks with the modulo function, we get,

$$\text{Size}_{\mathcal{O},f}(i) \triangleq |\{o_j : f(o_j) \mod k = i\}| \ .$$

We call a mapping f *balanced* if $\text{Size}_{\mathcal{O},f}(i) = \text{Size}_{\mathcal{O},f}(j)$ for every $i, j \in [k]$, otherwise it is called *unbalanced*[4].

5.2 Pairwise Information is may be Misleading

We start by constructing an instance, $(\mathcal{O}, \sigma)$, of the k-cache misses problem for which the optimum to the k-cache pairwise misses problem is a factor of $\frac{k-1}{2}$ away

[4]In our analysis, the number of objects m is always an integer multiplication of the number of sets in the cache, k. A natural generalization of this definition allows, when m is not a multiplication of k, a difference of at most 1 between the maximum and the minimum values of $\text{Size}_{\mathcal{O},f}(i)$ over all i's.

from the optimum of the k-cache misses problem. Later we will show how to alter this construction in order to get an instance for which the optimum to the k-cache pairwise misses problem is a factor of $k-3$ away from the optimum of the k-cache misses problem.

The instance contains k^2 memory objects, and an access sequence which treats them as k sets of k objects. Specifically, let $\mathcal{O} = \{o_{ij} : i \in [k], j \in [k]\}$ (recall that $[k] = \{0, \ldots, k\}$), and let $\sigma = \sigma_1 \bigodot \sigma_2$, where

$$\sigma_1 = \bigodot_{i=0}^{k-1} \left(\bigodot_{r=0}^{k-1} o_{ir} \right)^{3\ell}$$

$$\sigma_2 = \bigodot_{i=0}^{k-2} \bigodot_{j=i+1}^{k-1} \left(\bigodot_{r=0}^{k-1} \bigodot_{s=0}^{k-1} (o_{ir}, o_{js})^{2\ell} \right)$$

and ℓ is a large number (to be determined later). As before, any ordering of the sub-sequences suffices for the correctness of our results.

The intuition behind this construction is the following. The goal is to make the optimal pairwise placement be much different from the real optimal placement. This sequence makes sure that a pairwise optimal solution (i.e., an optimal solution for the k-cache pairwise misses problem), places the objects $o_{i0}, \ldots, o_{i(k-1)}$ at addresses corresponding to different cache sets for any i. On the other hand, the real optimal placement does the opposite: for each i, it places the objects $o_{i0}, \ldots, o_{i(k-1)}$ at addresses corresponding to the same cache line.

The goal of the first sequence, σ_1, is to mislead the pairwise target function. This sequence seems extremely expensive to the pairwise target function because there are $\binom{k}{2}$ pairs, each causing many cache misses. However, when the sequence is really executed, these costs are not disjoint and need not be counted $\binom{k}{2}$ times. After blinding the pairwise target function in this manner, we make it pay for its mistake using the second sequence σ_2. In the eyes of the pairwise target function the cost of σ_2 seems small comparing to the cost of σ_1. This is of course not true when the full sequence is checked. The optimal placement pays very little when executing σ_2 and this is how the gap between the two placements is obtained. We go on and formalize this intuition.

In what follows it is important to remember that ℓ is a large number, much larger than k. The access sequence σ consists of inner sub-sequences (containing k or 2 memory objects) which appear either 3ℓ or 2ℓ times. In σ_1 an inner sub-sequence is of the form $\bigodot_{r=0}^{k-1} o_{ir} = (o_{i0}, \ldots, o_{i(k-1)})$ and in σ_2 it is of the form (o_{ir}, o_{js}). Consider such an inner sub-sequence $\tilde{\sigma}$. Given a mapping f, we can divide the cache misses occurring during the repetitions of $\tilde{\sigma}$ into two types. The first type are misses occurring during the first time the sub-sequence $\tilde{\sigma}$ is accessed, and the second type are the misses that occur in the rest of the repetitions of $\tilde{\sigma}$. We consider this difference because we want to discuss separately the required cost of bringing some objects to the cache the first time they are accessed, and the (not always required) cost of repeatedly bringing them to the cache again and again (because the placement was not good). If $\tilde{\sigma}$ contains two or more objects that are mapped to addresses which correspond to the same cache line a miss occur in each repetition. Otherwise, no misses of the second type occur. The sequence σ_1

contains k inner sub-sequences of size k, and σ_2 contains $k^2\binom{k}{2}$ inner sub-sequences of size 2. Therefore, there can be up to $k^2(1+2\binom{k}{2}) \leq k^4$ misses due to the first type of misses for every possible mapping. As can be expected, misses of the second type will be the dominant in our analysis.

Our plan is as follows. We start with showing that the optimal placement has a relatively low cost. Next, we show that the placement that gets the pairwise target function to the minimum has a specific structure. Finally, we show that with this structure, the optimal pairwise placement must cause a lot of cache misses.

We fix $\ell = k^3$, and we start with an upper bound on the value of an optimal mapping for the k-cache misses problem.

Lemma 5.1. *Let f be an optimal placement for the k-cache misses problem. Then, Misses$((\mathcal{O},\sigma), f) \leq 4\ell k^2$.*

Proof. Let g be a placement that places the objects in the set $\mathcal{O}_i \triangleq \{o_{ij} : j \in [k]\}$ in memory locations corresponding to the i'th line of the cache. $\text{Misses}((\mathcal{O},\sigma_1), g) = 3\ell k^2$, and $\text{Misses}((\mathcal{O},\sigma_2), g) \leq k^4$. Thus, $\text{Misses}((\mathcal{O},\sigma), g) \leq 3\ell k^2 + k^4 \leq 4\ell k^2$. Since f is an optimal placement its cost is bounded by the cost of using g. $\square$

Consider two objects $o_{i_1j_1}, o_{i_2j_2} \in \mathcal{O}$, and denote by $\sigma_{(i_1j_1),(i_2j_2)}$ the pairwise sequence of the two objects, i.e., the projection of the sequence σ on these two objects. If $i_1 = i_2$ then $\sigma_{(i_1j_1),(i_2j_2)}$ consists of a prefix $(o_{i_1j_1}, o_{i_2j_2})^{3\ell}$ that is projected from σ_1 and a suffix that contains $(k-1)k$ copies of $o^{2\ell}_{i_1j_1}$ and $(k-1)k$ copies of $o^{2\ell}_{i_2j_2}$ projected from σ_2. Namely,

$$\begin{aligned}\sigma_{(i_1j_1),(i_1j_2)} = \; & (o_{i_1j_1}, o_{i_1j_2})^{3\ell} \bigodot \\ & (o^{2\ell}_{i_1j_1} \bigodot o^{2\ell}_{i_1j_2})^{k\cdot(i_1-1)} \bigodot \\ & ((o^{2\ell}_{i_1j_1})^k \bigodot (o^{2\ell}_{i_1j_2})^k)^{(k-i_1)}\end{aligned}$$

On the other hand, if $i_1 < i_2$ then $\sigma_{(i_1j_1),(i_2j_2)}$ consists of a prefix $o^{3\ell}_{i_1j_1} \bigodot o^{3\ell}_{i_2j_2}$ originating from σ_1, and three kinds of contributions from σ_2: a sub-sequence $(o_{i_1j_1}, o_{i_2j_2})^{2\ell}$, $k(k-1)-1$ copies of $o^{2\ell}_{i_1j_1}$, and $k(k-1)-1$ copies of $o^{2\ell}_{i_2j_2}$. That is,

$$\begin{aligned}\sigma_{(i_1j_1),(i_2j_2)} = \; & o^{3\ell}_{i_1j_1} \bigodot o^{3\ell}_{i_2j_2} \bigodot \\ & ((o^{2\ell}_{i_1j_1})^k \bigodot (o^{2\ell}_{i_2j_2})^k)^{(i_1-1)} \bigodot \\ & (o^{2\ell}_{i_2j_2})^{(j_1-1)} \bigodot (o^{2\ell}_{i_1j_1})^{(j_2-1)} \bigodot \\ & (o_{i_1j_1} \bigodot o_{i_2j_2})^{2\ell} \bigodot \\ & (o^{2\ell}_{i_1j_1})^{(k-j_2)} \bigodot (o^{2\ell}_{i_2j_2})^{(k-j_1)} \bigodot \\ & (o^{2\ell}_{i_2j_2})^{k\cdot(k-i_1-1)}\end{aligned}$$

The number of misses resulting from placing the two objects at addresses corresponding to the same cache line can be divided into two types. The first type of misses are those that occur due to the sub-sequence which contains either 2ℓ or 3ℓ repetitions of $(o_{i_1j_1}, o_{i_2j_2})$, and the second type are the rest. If we access the same object many times, it is as if we have accessed it only once, as far as the cache is concerned. Therefore, there are no more than $2k(k-1)$ pairwise misses of the second type (in both cases).

We now show that a pairwise optimal solution must be balanced.

LEMMA 5.2. *A placement f that minimizes Pairs$((\mathcal{O},\sigma),f)$ is balanced.*

PROOF. Let g be an unbalanced placement. We construct a (not necessarily balanced) placement f from g such that $\mathrm{Pairs}((\mathcal{O},\sigma),f) < \mathrm{Pairs}((\mathcal{O},\sigma),g)$. Thus, an unbalanced placement cannot be optimal with respect to the target function $\mathrm{Pairs}((\mathcal{O},\sigma),\cdot)$. We start by observing that because g is unbalanced there exist two cache sets i_1, i_2 for which $\mathrm{Size}_{\mathcal{O},g}(i_1) \geq k+1$ and $\mathrm{Size}_{\mathcal{O},g}(i_2) \leq k-1$. Namely, the number of objects that g maps to cache line i_1 is larger by at least 2 than the number of objects it maps to cache line i_2. Furthermore, there exists an index i for which g maps more objects from $\mathcal{O}_i = \{o_{ij} : j \in [k]\}$ to line i_1 than to line i_2. Denote by t_1 and t_2 the number of objects from $\mathcal{O}_i$ which are mapped to lines i_1 and i_2, respectively. Also, define $\Delta_r = \mathrm{Size}_{\mathcal{O},g}(i_r) - t_r$ for $i \in \{1,2\}$. Δ_1 and Δ_2 are the number of objects not from $\mathcal{O}_i$ that are mapped to the lines i_1 and i_2, respectively. Clearly,

$$(t_1+\Delta_1)-(t_2+\Delta_2) = \mathrm{Size}_{\mathcal{O},g}(i_1) - \mathrm{Size}_{\mathcal{O},g}(i_2) \geq 2\ . \qquad (1)$$

Let f be a placement, which places objects as g does, with the exception that one of the objects, which is mapped by g to line i_1, is mapped by f to line i_2. We begin the analysis by ignoring the second type of misses. We will later show that misses of the second type do not interfere too much with our calculations. We start by computing the difference in pairwise misses between g and f. Note that the difference originates only from misses in the i_1 and i_2 cache lines. Namely, due to the fact that t_1 is reduced by 1 and t_2 is increased by 1. We separate the calculation to σ_1 and σ_2. For σ_1 the modified cost is due to the interaction between the t_1 (or t_2) objects in the set $\mathcal{O}_i$:

$$\begin{aligned}\mathrm{Pairs}((\mathcal{O},\sigma_1),g) - \mathrm{Pairs}((\mathcal{O},\sigma_1),f) &= 6\ell(\tbinom{t_1}{2}+\tbinom{t_2}{2}-\tbinom{t_1-1}{2}-\tbinom{t_2+1}{2})\\ &= 6\ell(t_1-t_2-1)\end{aligned}$$

For σ_2 the main cost is the interaction of the moved object with the Δ_1 or Δ_2 objects that are not in $\mathcal{O}_i$:

$$\mathrm{Pairs}((\mathcal{O},\sigma_2),g) - \mathrm{Pairs}((\mathcal{O},\sigma_2),f) = 4\ell(\Delta_1-\Delta_2)\ .$$

We show that $\mathrm{Pairs}((\mathcal{O},\sigma),g) > \mathrm{Pairs}((\mathcal{O},\sigma),f)$ in each of three possible cases:

(1) $t_1 - t_2 = 1$.
In this case, $\Delta_1 - \Delta_2 \geq 1$ due to Equation 1. Thus, $\mathrm{Pairs}((\mathcal{O},\sigma_1),g) = \mathrm{Pairs}((\mathcal{O},\sigma_1),f)$, and $\mathrm{Pairs}((\mathcal{O},\sigma_2),g) > \mathrm{Pairs}((\mathcal{O},\sigma_k^2),f)$.

(2) $t_1 - t_2 \geq 2$ and $\Delta_1 - \Delta_2 \geq 0$.
$\mathrm{Pairs}((\mathcal{O},\sigma_1),g) > \mathrm{Pairs}((\mathcal{O},\sigma_1),f)$, and $\mathrm{Pairs}((\mathcal{O},\sigma_2),g) \geq \mathrm{Pairs}((\mathcal{O},\sigma_2),f)$.

(3) $t_1 - t_2 \geq 2$ and $\Delta_1 - \Delta_2 < 0$.
In this case, using Equation 1,

$$\begin{aligned}\mathrm{Pairs}((\mathcal{O},\sigma_1),g) - \mathrm{Pairs}((\mathcal{O},\sigma_1),f) &\geq 6\ell\cdot(2-\Delta_1+\Delta_2-1)\\ &= 6\ell\cdot(1-\Delta_1+\Delta_2)\ ,\end{aligned}$$

whereas,

$$\text{Pairs}((\mathcal{O},\sigma_2),g) - \text{Pairs}((\mathcal{O},\sigma_2),f) \geq 4\ell(\Delta_1 - \Delta_2) \ .$$

Thus,

$$\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq 6\ell(1 - \Delta_1 + \Delta_2) + 4\ell(\Delta_1 - \Delta_2) \ .$$

Using the fact that $\Delta_2 - \Delta_1 > 0$, we get that

$$\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq 6\ell > 0 \ .$$

We conclude that, $\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq \ell$. Now we must take into account the change in the first type of pairwise misses. As said earlier, there can be up to $2k(k-1)$ misses of the second type. Thus, in the worst case, $\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq k^3 - 2k(k-1) > 0$. (Recall that we fixed $\ell = k^3$.) This means that f is a better placement than g with respect to the k-cache pairwise problem. It follows that for every unbalanced placement there exist another better placement. Therefore, an optimal placement must be balanced. □

Next, we present a lower bound on the value of a pairwise optimal solution.

LEMMA 5.3. *Let f be an optimal pairwise placement (i.e., a placement that minimizes Pairs$((\mathcal{O},\sigma),f)$). Then, Misses$((\mathcal{O},\sigma),f) \geq 2\ell k^2(k-1)$.*

PROOF. Let g be a balanced placement. Consider two objects $o, o' \in \mathcal{O}$ which are placed in the same cache set by g. If $o, o' \in \mathcal{O}_i = \{o_{ij} : j \in [k]\}$ for some $i \in [k]$, then their contribution to $\text{Pairs}((\mathcal{O},\sigma),g)$ is at least 6ℓ misses (due to σ_1), otherwise they contribute no more than $4\ell + 2k(k-1)$ misses. $4\ell + 2k(k-1) < 6\ell$, thus, the best pairwise balanced placement f would be a placement which places the objects from $\mathcal{O}_i$ in different cache sets for any $i \in [k]$ (e.g., the one that places $o_{1j}, \ldots, o_{kj}$ in the same cache set for any j). Thus, due to σ_2, $\text{Pairs}((\mathcal{O},\sigma),f) \geq 4\ell k\binom{k}{2}$. However, recall that we want to find out the number of "real" misses that occur when using the placement f. To do that examine σ_2. As mentioned above, f places the objects from $\mathcal{O}_i$ in different cache sets for any $i \in [k]$. Thus, each object from $\mathcal{O}_i$ shares a cache set with objects that are not members of $\mathcal{O}_i$ for any $i \in [k]$. In other words, this means that all the $\mathcal{O}_i$'s are represented in every cache line C_r ($r \in [k]$). Consider a cache line C_r. There are k objects that are mapped to addresses corresponding to C_r, such that every pair appears in 2ℓ times in σ_2. Thus, the contribution of C_r to $\text{Misses}((\mathcal{O},\sigma),f)$ is $\binom{k}{2}4\ell$ misses. This is true for all cache lines, and, therefore, $\text{Misses}((\mathcal{O},\sigma),f) \geq 4\ell k\binom{k}{2} = 2\ell k^2(k-1)$. □

Finally, we can compare the values of an optimal mapping and an optimal pairwise mapping.

THEOREM 5.4. *For every $k \geq 2$ there exist an instance for which the number of misses due to an optimal pairwise mapping is a factor of $\frac{1}{2}(k-1)$ away from an optimal mapping.*

PROOF. Let f and g be an optimal placement and an optimal pairwise placement, respectively. By Lemma 5.1, $\text{Misses}((\mathcal{O},\sigma),f) \leq 4\ell k^2$. On the other hand, by Lemmas 5.2 and 5.3 $\text{Misses}((\mathcal{O},\sigma),g) \geq 2\ell k^2(k-1)$. The theorem follows. □

This result can be improved to a factor of $k-3$ in the following manner.

THEOREM 5.5. *For every $k \geq 3$ there exist an instance for which the number of misses due to an optimal pairwise mapping is a factor of $k-3$ away from an optimal mapping.*

PROOF. Replace the 3ℓ in σ_1 by $(k-2)\ell$, and the 2ℓ in σ_2 by $(k-3)\ell$. Let f and g be an optimal placement and an optimal pairwise placement, respectively. By slightly changing Lemma 5.1 we can show that $\text{Misses}((\mathcal{O},\sigma), f) \leq (k-2)\ell k^2 + k^4$. Using the fact that $\ell = k^3$, we get that $\text{Misses}((\mathcal{O},\sigma), f) \leq \ell k(k-1)^2$. We can also modify Lemmas 5.2 and 5.3 in order to prove that $\text{Misses}((\mathcal{O},\sigma), g) \geq (k-3)\ell k^2(k-1)$. The theorem follows from the fact that $\frac{k(k-3)}{k-1} \geq k-3$. $\square$

5.3 Using Pairwise Information is Hard

We have shown that there are instances for which the (real) value of a pairwise optimal placement is far from the value of an optimal placement. This means that in a worst case scenario a pairwise optimal placement may turn out to be a bad placement with respect to the k-cache misses problem. However, typical instances may behave otherwise. Therefore, one may benefit from finding a placement that exhibits good behavior with respect to pairwise information. Also, suppose that due to the cost of using the whole access sequence one must rely on pairwise information.

A natural question arises: can one efficiently find a solution that is good in the pairwise information setting? Namely, we are now looking for a solution that behaves well with respect to pairs, even if it behaves poorly with respect to the whole sequence. We show that, unless P = NP, there is no efficient optimal algorithm that minimizes the number of cache misses in the pairwise information setting. Moreover, we show that one cannot find an approximate placement whose value is within a factor of $O(n^{1-\varepsilon})$ from the optimal value.

In section 4.2 we have shown that for any $0 < \varepsilon \leq 1$ the k-cache misses problem cannot be approximated within $O(n^{1-\varepsilon})$. This bound was achieved by using a reduction from the k-colorability problem. We use the same reduction to show our hardness result for the k-cache pairwise misses problem. Given a graph G, we have constructed (in section 4.2) an instance $H_\ell(G) = (\mathcal{O}_G, \sigma_G)$ in which an edge $e = (u, v)$ was represented by a subsequence that contained many repetitions of two objects corresponding to the vertices u and v. Given a k-coloring c we have defined (in section 4.2) a family of placements F_c. Then, in Lemma 4.2, we have shown that a correlation between c and F_c exists. The following lemma shows a similar correlation with respect to pairwise information.

LEMMA 5.6. *For any k-coloring c of V, and for any $f \in F_c$,*

$$Pairs(H_\ell(G), f) = \Theta(|E|^\ell\, Mono(G, c)) + O(|E|) \ .$$

PROOF. Consider a mapping $f \in F_c$. Examine two objects o_i, o_j, and the misses caused by $\sigma_{i,j}$. First, let v_i and v_j be the corresponding vertices in G. Now, examine an edge (v_i, v_j). If (v_i, v_j) is monochromatic then $f(o_i)$ and $f(o_j)$ are mapped to the same cache line. Therefore, the sub-sequence $(o_i, o_j)^{|E|^\ell}$ that corresponds to a monochromatic edge contributes at least $2\,|E|^\ell - 1$ misses. On the other hand, if (v_i, v_j) is not monochromatic then $f(o_i)$ and $f(o_j)$ are not mapped to the same cache line. Thus, a non-monochromatic edge causes no more than two misses. Furthermore, the subsequences $o_i^{|E|^\ell}$ and $o_j^{|E|^\ell}$ appear up to $\deg(o_i)$ and $\deg(o_j)$ times in $\sigma_{i,j}$, respectively, where $\deg(u)$ is the number of edges incident on u. Thus,

$$\begin{aligned}
\mathrm{Pairs}(H_\ell(G), f) &= \sum_{i<j} \mathrm{Misses}((\{o_i, o_j\}, \sigma_{i,j}), f) \\
&= \Theta(|E|^\ell \,\mathrm{Mono}(G, c)) + \sum_{i<j} (\deg(v_i) + \deg(v_j)) \\
&= \Theta(|E|^\ell \,\mathrm{Mono}(G, c)) + O(|E|) \ .
\end{aligned}$$

□

The following theorem and corollary can be proven using similar arguments to those used to prove Theorem 4.4, and Corollary 4.5 in Section 4.

THEOREM 5.7. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the k-cache pairwise misses problem cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

COROLLARY 5.8. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the k-cache pairwise misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

6. APPROXIMATION ALGORITHMS

In light of the harsh lower bounds it is clear that an algorithm that works well for all access sequences does not exist. However, it is still interesting to know whether an algorithm with a mild approximation ratio exists, or, whether the hardness results can be improved. In this section we show that the hardness results established in Sections 4 and 5.3 are essentially tight. Specifically, we present $\frac{n}{c \log n}$-approximation algorithms, for any $c > 0$, for the problems discussed in this paper. Our algorithms are constructed in the spirit of the approximation algorithm for the 3-partition problem from [Kann et al. 1997]. In the following theorems we show that if the optimum is high then any placement can be used as an approximate solution. On the other hand, if the optimum is low then an optimal solution can be found in polynomial time.

THEOREM 6.1. *For every $k \geq 2$ and $c > 0$ the k-cache misses problem can be approximated within $\frac{n}{c \log n}$ in polynomial time.*

PROOF. Consider an instance I of the k-cache misses problem. If $\mathrm{Opt}(I) > c \log n$ then the value of any placement is not greater than $\frac{n}{c \log n}$ times the optimum. This is because $\mathrm{Misses}(I, f) \leq n$ for every placement f. Assume that $\mathrm{Opt}(I) \leq c \log n$. A cache miss must occur when an object is accessed for the first time, therefore, the optimum number of misses is not less than the number of objects in the access sequence. Thus, there are no more than $c \log n$ objects in the access sequence. We now show that in this case an optimal solution can be found in polynomial time. It is sufficient to examine $k^{c \log n} = n^{c \log k}$ placements in order to find an optimal placement. Moreover, the number of misses due to a candidate placement can be calculated in polynomial time by simply executing the sequence. Thus, an $\frac{n}{c \log n}$-approximate solution can be found in polynomial time by the following algorithm. First, check whether the number of objects in the access sequence is greater than $c \log n$. If it is, output an arbitrary placement. Otherwise, find an optimal solution as described above. $\square$

The following can be proven similarly.

THEOREM 6.2. *For every $k \geq 2$ and $c > 0$ the t-way k-cache misses problem can be approximated within $\frac{n}{c \log n}$ in polynomial time.*

THEOREM 6.3. *For every $k \geq 2$ and $c > 0$ the k-cache M-memory misses problem can be approximated within $\frac{n}{c \log n}$ in polynomial time.*

THEOREM 6.4. *For every $k \geq 2$ and $c > 0$ the k-cache pairwise misses problem can be approximated within $\frac{n}{c \log n}$ in polynomial time.*

Since $\log k$ is around 10 in practice, the time complexity of these algorithms is far from being practical. However, these upper bounds are interesting since they indicate that the lower bounds presented in this paper are essentially tight.

7. ALLOWING UNRESTRICTED ALIGNMENTS

Until now we have assumed that the memory objects have the same size – the size of a cache line. We have also assumed that each object is placed in memory such that only one cache line (or set) will be involved when it is accessed. That is, each object is aligned and sized to fit one complete cache line.

The first assumption is reasonable when dealing with lower bounds, because a lower bound on the uniform size version of the problem implies the same lower bound on the more general non uniform size version of the problem. On the other hand, such an assumption is not reasonable when constructing an approximation algorithm. That is, an approximation algorithm for the uniform size case does not imply an approximation algorithm for the more general case. However, it turns out that our approximation algorithms can be extended to the case where object sizes are not necessarily equal to the size of a cache line. Specifically, we show that arguments similar to those given in the proof of Theorem 6.1 work for the more general case as well.

Under the second assumption, a mapping must place objects such that they are aligned. We show that our approximation algorithms can be modified to work without the this assumption. However, our negative results are not easily extended to the unrestricted alignments setting. When objects are forced to be aligned many feasible solutions are discarded, and, therefore, the optimum may change. In fact, there are instances for which the optimum does change as demonstrated in the following example. Consider a cache with three lines each of size L. Let $\mathcal{O} = \{o_1, o_2\}$, where o_1 and o_2 are of size $3L/2$, and let $\sigma = (o_1, o_2)^\ell$. If we must place objects at the beginning of memory blocks, then these two objects must collide, and, therefore, $\text{Misses}((\mathcal{O}, \sigma), f) \geq 2\ell$ for any mapping f. On the other hand, if f places the two objects consecutively in the memory, i.e., such that at least one object is not aligned, we get that $\text{Misses}((\mathcal{O}, \sigma), f) = 3$. In this example there are objects that are not of size L, and this is not a coincidence. It turns out that if all objects are of size L (cache line size), then the use of placements of unrestricted alignments does not help much. Specifically, we show that, when all objects are of size L, for every placement f that is not aligned there exists an aligned placement whose value is not much worse than the value of f. Thus, our negative results can be extended to the unrestricted alignments setting. In particular, we show that our lower bounds remain valid, and that the use of pairwise information may produce a bad placement.

7.1 Direct-Mapped Caches

We start by formally redefining the problem with unrestricted alignments for the case of direct mapping. The t-way set-associative, finite memory, and pairwise versions can be redefined similarly. As before, we assume the size of the memory is not bounded. Let L be the size of a cache line. An address $N = N' \cdot L + l$ in the memory is assigned by the direct mapping to the cache line $r = (N' \bmod k)$, and to the offset $l = (N \bmod L)$ within the cache line r. Given an input $(\mathcal{O}, \sigma)$ the goal is to find a memory placement of the objects in $\mathcal{O}$ that minimizes the number of cache misses which occur when accessing the data objects according to the sequence σ. As before we use $\text{Misses}((\mathcal{O}, \sigma), f)$ to denote the number of cache misses occurring while accessing the memory according to a sequence σ of objects from $\mathcal{O}$. Note that the input remains in the same shape, and so does the objective function. The only difference is in the placement function f. We formalize the above in the following manner:

Minimum unrestricted alignments k-cache misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N} \times [L]$.

Measure: $\text{Misses}((\mathcal{O}, \sigma), f)$.

An instance of the unrestricted alignments k-cache misses problem is called *uniform* if the size of any object is L. Given an instance to the unrestricted alignments k-cache misses problem, a placement f is called *aligned* if the offset within the cache line is zero for every object, i.e., $f(o) = (N_o, 0)$ for some $N_o \in \mathbb{N}$ for every object $o \in \mathcal{O}$. If there exists an object $o \in \mathcal{O}$ for which $f(o) = (N_o, l)$ and $l \neq 0$ then both the placement f and the object o are referred to as *non aligned*. Note that in the previous sections we assumed that all instances are uniform and that all placements are aligned.

Consider a uniform instance. When accessed, aligned objects are put in one cache line. On the other hand, non aligned objects are put in two consecutive (modulo k) cache lines. Note that sometimes a non aligned object o can be found in the cache due to accesses to objects that are placed before and after o. Clearly, this distinction does not exist where an aligned placement is concerned. We show that one can assume that all placements are aligned when dealing with uniform instances. Specifically, in the next lemma, we prove that for any non aligned placement f there exists an aligned placement g which does not cause more misses than f. This implies that the use of non aligned placements does not help when dealing with uniform instances. That is, there exists an aligned optimal placement in the case of uniform instances.

LEMMA 7.1. *Let $(\mathcal{O}, \sigma)$ be a uniform instance. For any non aligned placement f there exists an aligned placement g such that $Misses((\mathcal{O}, \sigma), g) \leq Misses((\mathcal{O}, \sigma), f)$.*

PROOF. Consider a uniform instance $(\mathcal{O}, \sigma)$ and a non aligned placement f. We show that there exists a placement f' such that $\text{Misses}((\mathcal{O}, \sigma), f') \leq \text{Misses}((\mathcal{O}, \sigma), f)$ and $n(f') \leq n(f)$, where $n(f)$ and $n(f')$ are the number of non aligned objects with respect to f and f', respectively. Since $n(f)$ is finite this proves the lemma.

Consider the objects in $\mathcal{O}$ in increasing order of memory addresses to which they were assigned by the placement f, and denote by o' the non aligned object with minimum address. Let $f(o') = (N, l)$, i.e., f places o' in the Nth memory block with offset l. Note that o' is the only object that is placed in the Nth memory block, since it is the non aligned object with minimum address. Moreover, after an access to o' cache lines $r = (N \bmod k)$ and $r' = (N + 1 \bmod k)$ contain memory blocks N and $N + 1$, respectively. Define the placement f' as follows:

$$f'(o) = \begin{cases} f(o) & o \neq o', \\ (N, 0) & o = o'. \end{cases}$$

That is, f' is exactly the same as f except that f' places o' at address $N \cdot k$ instead of at $N \cdot k + l$. Clearly, $n(f') = n(f) - 1$. Also, after an access to o' cache line r contains the Nth memory block, since f' places o' in memory block N. Furthermore, since o' is the only object that is placed in the Nth memory block by f, the remapping of object o' may only influence the number of misses involving cache line r'.

We now turn to prove that f' is not worse than f. First, consider the case in which no other object is placed by f at memory block $N + 1$. In this case, no object is placed by f' at memory block $N + 1$, and therefore when using f' there are no accesses to memory block $N + 1$. Since o' is the only object that is placed at memory blocks N and $N + 1$ by f, a cache miss due to object o' involving cache line r with respect to f' translates to at least one miss with respect to f—one miss at line r, and possibly another at line r'. Also, when using f, there may be more misses that are caused by the presents of block $N + 1$ in cache line r'. Thus, $\text{Misses}((\mathcal{O}, \sigma), f') \leq \text{Misses}((\mathcal{O}, \sigma), f)$.

Consider the case where another object, denoted by o'', is placed at memory block $N + 1$ by f, and hence also by f'. o'' is non aligned, because f places a part of o' in the beginning of memory block $N + 1$. Thus, when brought into the cache, object o'' is put in cache lines r' and $(r' + 1 \bmod k)$. As mentioned before, the remapping of o' does not influence the misses involving any cache line $q \neq r'$, thus we examine the misses involving cache line r'. The only possible scenario in which a miss occurs with respect to f' and not with respect to f is when o'' is accessed and block $N + 1$ is already in the cache due to a some previous access to o'. This previous access caused a miss due to memory block $N + 1$ with respect to f, but not with respect to f'. Namely, the miss simply occurred earlier under f. Therefore, any miss with respect to f' can be assigned to a unique miss with respect to f. And this means that $\text{Misses}((\mathcal{O}, \sigma), f') \leq \text{Misses}((\mathcal{O}, \sigma), f)$. $\square$

By Lemma 7.1 there exists an aligned optimal solution for uniform instances. Therefore, a lower bound on aligned placements serves as lower bound on all placements. Thus, we can extend Theorem 4.4 and Corollary 4.5 to the case of unrestricted alignments.

THEOREM 7.2. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the unrestricted alignments k-cache pairwise misses problem cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

COROLLARY 7.3. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the unrestricted alignments k-cache pairwise misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Lemma 7.1 was stated for the direct mapping case, but it can be also proven for the finite memory, and pairwise cases. It is not hard to see that Lemma 7.1 remains valid for the k-cache M-memory misses problem. We now turn to the pairwise case. Recall that, in this case, given an instance $(\mathcal{O}, \sigma)$ we sum up the misses in all pairwise sequences $\sigma_{i,j}$ for $o_i, o_j \in \mathcal{O}$. Lemma 7.1 is valid for all sequences, and, in particular, for pairwise sequences. (Indeed, the placement f is shared by all pairwise sequences. However, the aligned placement g is the same for all pairwise sequences as well.) Thus, Lemma 7.1 can be restated for the pairwise case. We conclude that the hardness results from section 4.4 and 5.3 can be extended to the unrestricted alignments case. Furthermore, Theorem 5.5 can be restated in the unrestricted alignments setting. That is, the instances we have constructed in Section 5.2 can be still used as evidence to a multipletive gap of $k - 3$ between an optimal pairwise mapping and an optimal mapping (in the worst case).

7.2 t-way Set-Associative Caches

Finally, we consider placements with unrestricted alignments in the case of t-way set-associative caches. Recall that in this case a replacement protocol is involved in deciding which object is to be removed from the cache when a miss occurs. A logical course of action would be to prove a version of Lemma 7.1 for t-way set-associative caches. That is, we would like to prove that given a uniform instance, for any placement f, the placement g we get by changing all offsets to zero (as was done in the proof of Lemma 7.1) is not worse than f. However, it turns out that such the lemma cannot be restated for t-way set-associative caches. For example, consider a 2-way set-associative cache whose replacement protocol is LRU, the instance $I = (\{o_1, o_2, o_3, o_4\}, (o_2, o_3, o_1, o_4, o_2, o_3, o_1, o_4, o_2))$, and a placement f for which $f(o_1) = (1, 1)$, $f(o_2) = (2, 1)$, and $f(o_3) = f(o_4) = (2, 0)$. It is not hard to verify that $\text{Misses}_2(I, f) = 7$, and $\text{Misses}_2(I, g) = 8$. The reason for this strange scenario is that LRU's decision with respect to a certain object may be influenced by an access to another object which resides in the same memory block, and consequently in the same cache block. So we need to prove a weaker version of the lemma. Another difficulty stems from the fact that the only information we have on the replacement protocol is that it is sensible.[5] Different replacement protocols

[5] It would have been natural to redefine the notion of sensibility in the unrestricted alignments setting. However, we only require sensibility when dealing with uniform instances and aligned placements.

may choose to remove different objects, and this may have a dramatic effect on the contents of the cache at any given moment. One possibility to tackle this difficulty is to prove that the use of unrestricted alignments does not help much for specific replacement protocols. However, we prefer to go about it differently. That is, instead of trying to prove weaker versions of the lemma for specific replacement protocols, we prove a much weaker property for all t-way set-associative caches that use sensible replacement protocols. We show that for some sequences, the number of misses does not increase by much after changing a placement of unrestricted alignments f to an aligned placement g by placing all objects in the beginning of the memory block in which they were originally placed (as formally defined in the proof of Lemma 7.1).

Consider a sequence of the form τ^{ℓ}. We show that if during each access to the subsequence τ at least one miss occurs when using the placement g then at least one miss occurs when we use f. Recall that when the replacement protocol is sensible then if the objects are placed in memory by g such that not more than t of them are mapped to the same cache set then only $O(1)$ misses occur. Thus, if a miss occurs during each access to τ then more than t objects are mapped to some set j, or, in other words, are placed in memory by g in the beginning of blocks corresponding to set j. By the construction of g, we know that these objects are placed in memory by f somewhere (i.e., with some offset) in blocks corresponding to set j. Therefore, at least one miss must occurs during each access to τ when using f.

Now, consider a sequence σ that can be divided into a series of sequences each composed of many iterations of some short subsequence, that is, $\sigma = \bigodot_{i=1}^{p} \tau_i^{n_i}$, where $|\tau_i| = O(1)$. We show that $\text{Misses}((\mathcal{O}, \sigma), g) = O(\text{Misses}((\mathcal{O}, \sigma), f)) + O(q)$. Due to the previous discussion, if at least one miss must occurs during each access to τ_i when using g then at least one miss must occurs during each access to τ_i when using f. Also, during the first few times we are accessing τ_i the placement g may cause more misses than f due to the different cache configuration at any given moment. This may result in an additional $O(q)$ misses because $|\tau_i| = O(1)$ for every i.

Examine the structure of sequence σ_G that was defined in Section 4.3. It is composed of a series of $|E|$ subsequences of length $k(t-1)+2 = O(1)$ that are accessed $|E|^{\ell}$ times, and a series of t subsequences of length $kt = O(1)$ that are accessed $|E|^{\ell+2}$ times. Thus, σ_G satisfies the above mentioned conditions, which means that $\text{Misses}((\mathcal{O}, \sigma_G), g) = O(\text{Misses}((\mathcal{O}, \sigma_G), f) + O(|E|)$ for any placement f of unrestricted alignments, where g is is the aligned placement resulting from placing all objects in the beginning of the memory block in which they were placed by f. Thus, a lower bound on all aligned placements implies a similar lower bound on all placements up to an additive factor of $O(|E|)$. We conclude that Proposition 4.7 can be restated in the unrestricted alignments setting, and, consequently, we can our lower bound remains valid in this setting.

THEOREM 7.4. *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the unrestricted alignments t-way k-cache misses problem (when using a sensible replacement protocol) cannot be efficiently approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

COROLLARY 7.5. *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimal value of the unrestricted alignments t-way k-cache misses problem (when using a sensible re-*

placement protocol) cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, *unless* $P = NP$.

We could have used similar arguments in order to show that the lower bounds for direct mapping hold even if non aligned placements are allowed. Instead, we have proven a much stronger relation between non aligned placements and aligned placements (Lemma 7.1). Note that this relation was essential in proving that there exists a multipletive gap of $k-3$ between an optimal pairwise mapping and an optimal mapping (in the worst case).

7.3 Approximation Algorithms

In this section we present approximation algorithms that work in the unrestricted alignments setting. As before, due to the harsh lower bounds, the approximation ratios of the algorithms are mild. We start by limiting our discussion to uniform instances, and then proceed to deal with the more general problem.

The next theorem shows that our approximation algorithm for the k-cache misses problem (see Section 6) can be used for the unrestricted alignments k-cache misses problem in the case of uniform instances. Similar theorems can be stated for the finite memory, and pairwise cases.

THEOREM 7.6. *For every* $k \geq 2$ *and* $c > 0$ *the unrestricted alignments* k*-cache misses problem can be efficiently approximated within* $\frac{n}{c \log n}$ *in the case of uniform instances.*

PROOF. As mentioned earlier, due to Lemma 7.1, we know that there exist an optimal aligned placement in the case of uniform instances. The rest of the proof is the same as the proof of Theorem 6.1. □

What about non uniform instances? A natural course of action is to try to construct an approximation algorithm that is similar to the one that was presented for the aligned case (in Section 6). That is, given an instance I, it is natural to try to show that if $\mathrm{Opt}(I)$ is large enough, then any placement is a "reasonable" placement, and that, otherwise, an optimal placement can be found in polynomial time. (Recall that in the aligned case we have found an optimal placement by checking a polynomial size set of placements that represents all possible placements.) However, this is easier said than done. This is because a non-uniform instance may contain objects of size less than L. For example, consider two objects of size $\frac{1}{2}L$ that are placed at addresses corresponding to the same cache block, and such that one is placed with offset 0 and the other with offset $\frac{1}{2}L$. Both objects may be placed in different memory blocks, or in the same memory block. Clearly, the number of misses may change dramatically according to the way we have chosen to place the objects. Thus, a reasonable set of placements that represents all possible placements must contain a placement for any possible grouping of objects. Such a set may be of super polynomial size when $\mathrm{Opt}(I) = \Theta(\log n)$. However, when $\mathrm{Opt}(I) \leq c$, for some constant c, the size of the set is $O(1)$, and, therefore, an optimal placement can be found in polynomial time. We show this in the proof of the next theorem. Similar theorems can be proven for the the t-way set-associative, finite memory, and pairwise cases.

THEOREM 7.7. *For every $k \geq 2$ and $c > 0$ the unrestricted alignments k-cache misses problem can be efficiently approximated within $\frac{n}{c}$.*

PROOF. Consider an instance $I = (\mathcal{O}, \sigma)$ of the unrestricted alignments k-cache misses problem. We assume (without loss of generality) that all the objects in $\mathcal{O}$ are accessed by σ. Let $|o|$ be the size of an object $o \in \mathcal{O}$ in terms of blocks. Each object must be brought into the cache when accessed, and, therefore, $\mathrm{Misses}(I, f) \geq \sum_{o \in \mathcal{O}} |o|$ for any placement f, or, in other words, $\mathrm{Opt}(I) \geq \sum_{o \in \mathcal{O}} |o|$. $|o| \geq \frac{1}{L}$ for any $o \in \mathcal{O}$. (Recall that the size is a cache block is L.) Thus, $\mathrm{Opt}(I) \geq |\mathcal{O}|/L$. On the other hand, an access to an object o can cause up to k misses. (We assume that an object is not larger than the cache.) Thus, $\mathrm{Opt}(I) \leq k \cdot n$. If $\mathrm{Opt}(I) > c \cdot k$ then any placement is an $\frac{n}{c}$ approximation. Otherwise, $|\mathcal{O}|/L \leq \mathrm{Opt}(I) \leq c \cdot k$. Which means that $|\mathcal{O}| \leq c \cdot k \cdot L$. We now show that in this case an optimal solution can be found in polynomial time.

The first problem we face is that an object can be placed anywhere in the memory, and, in particular, not necessarily such that it is aligned. Thus, we must try to place an object not only at addresses corresponding to all k cache lines, but also at addresses corresponding to all alignments within the lines. Also, several objects may share the same memory block, so simply finding a memory block that corresponds to a specific cache line and a specific offset is not enough. To cope with these problems we try all possible placements of the objects in the first $k|\mathcal{O}|$ blocks of memory. By doing this we assign each object to addresses corresponding to any line and offset in this line. Also, all possible groupings are covered. Thus, it is sufficient to examine $(kL|\mathcal{O}|)^{|\mathcal{O}|} \leq (ck^2L^2)^{ckL}$ placements in order to find an optimal placement. Moreover, the number of misses due to a candidate placement can be calculated in polynomial time by simply executing the sequence. Thus, an $\frac{n}{c}$-approximate solution can be found in polynomial time by the following algorithm. First, check whether the number of objects in the access sequence is greater than $c \cdot k \cdot L$. If it is, output an arbitrary placement. Otherwise, find an optimal solution as described above. □

8. CONCLUSION

We have studied an important challenge in todays computing environment: how to place data and code in the memory so that cache misses are reduced to minimum. We have shown that in the worst case, an efficient algorithm cannot even come close to achieving this goal (not even just estimate the number of misses for the optimal placement), unless P = NP. Thus, this problem joins the small family of extremely inapproximable optimization problems.

We have also studies the question of how much information is lost when one keeps only pairwise relations between the objects. We have shown that in the worst case critical information is lost in this manner, and that it is not possible to get close to the optimal memory arrangement even if one has no time limits. Furthermore, we have shown that finding a placement that is good with respect to pairwise information is computationally hard as the original problem.

Finally, we have presented approximation algorithms for these problems. The algorithms have mild approximation ratios, but we cannot hope to do much better in light of the harsh lower bounds we have shown.

The implication of the first result is that we must use heuristics. There is no hope to find one (efficient) algorithm that does well for all possible benchmarks (unless P = NP). One should study how normal programs behave and do well with respect to them. Another important implication is that it is not possible to evaluate the potential of reducing cache misses via such methods. Thus, one cannot tell, before implementing a cache conscious data (or code) placement algorithm, what the potential benefit is. Also, one cannot evaluate how much of the potential benefit the implemented algorithm actually obtains. The measure of such algorithms should be their improvement over existing non-cache-conscious algorithms on given benchmarks.

The implication of the second result (about keeping information for pairs) is that if one wishes to spend a long time finding the optimal arrangement in the memory via a full search, then one should be careful about the way data is processed. Keeping information in a pairwise manner is not good in this case: even if one has infinite time resources, pairwise information is not enough to even get close to the optimal solution.

ACKNOWLEDGMENT

We thank Seffi Naor for helpful discussions.

REFERENCES

Arora, S. and Lund, C. 1997. Hardness of approximations. See Hochbaum [1997], Chapter 10, 399–446.

Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. 1998. Proof verification and the hardness of approximation problems. *Journal of the ACM 45,* 3, 501–555.

Arora, S. and Safra, S. 1998. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM 45,* 1, 70–122.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., and Protasi, M. 1999. *Complexity and Approximation; Combinatorial optimization problems and their approximability properties.* Springer Verlag.

Belady, L. A. 1966. A study of replacement algorithms for virtual-storage computers. *IBM Systems Journal 5,* 2, 78–101.

Bellare, M., Goldreich, O., and Sudan, M. 1998. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM J. Comput. 27,* 3, 804–915.

Boehm, H.-J. 2000. Reducing garbage collector cache misses. In *2nd International Symposium on Memory Management.* 59–64.

Boppana, R. and Halldórsson, M. M. 1992. Approximating maximum independent sets by excluding subgraphs. *BIT 32*, 180–196.

Calder, B., Krintz, C., John, S., and Austin, T. 1998. Cache-conscious data placement. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems.* 139–149.

Chilimbi, T. M., Davidson, B., and Larus, J. R. 1999. Cache-conscious structure definition. In *ACM SIGPLAN Conference on Programming Languages Design and Implementation.* 13–24.

Chilimbi, T. M., Hill, M. D., and Larus, J. R. 1999. Cache-conscious structure layout. In *ACM SIGPLAN Conference on Programming Languages Design and Implementation.* 1–12.

Chilimbi, T. M. and Larus, J. R. 1998. Using generational garbage collection to implement cache-conscious data placement. In *1st International Symposium on Memory Management.* 37–48. ISMM is the successor to the IWMM series of workshops.

DING, C. AND KENNEDY, K. 1999a. Improving cache performance in dynamic applications through data and computation reorganization at run time. In *ACM SIGPLAN Conference on Programming Languages Design and Implementation*. 229–241.

DING, C. AND KENNEDY, K. 1999b. Inter-array data regrouping. In *Workshop on Languages and Compilers for Parallel Computing*.

FEIGE, U. 1998. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM 45,* 4, 634–652.

FEIGE, U., GOLDWASSER, S., LOVASZ, L., SAFRA, S., AND SZEGEDY, M. 1991. Approximating clique is almost NP-complete. In *32th IEEE Symposium on Foundations of Computer Science*. 2–12.

FEIGE, U. AND KILIAN, J. 1998. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci. 57,* 2, 187–199.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 41 Madison Avenue, New York, NY 10010.

GLOY, N. AND SMITH, M. D. 1999. Procedure placement using temporal-ordering information. *ACM Trans. Program. Lang. Syst. 21,* 5, 977–1027.

HALLDÓRSSON, M. M. 1993. A still better performance guarantee for approximate graph coloring. *Inf. Process. Lett. 45,* 1, 19–23.

HÅSTAD, J. 1996. Clique is hard to approximate within $n^{1-\epsilon}$. In *37th IEEE Symposium on Foundations of Computer Science*. 627–636.

HENIS, E. A., HABER, G., KLAUSNER, M., AND WARSHAVSKY, A. 1999. FDPR - a post-link optimization tool for large subsystems. *Feedback Directed Optimizations 2 Workshop*.

HENNESSY, J. L. AND PATTERSON, D. A. 1996. *Computer Architecture—A Quantitative Approach*, Second ed. Morgan Kaufmann Publishers, San-Francisco, CA 94104-3205, USA.

HOCHBAUM, D. S., Ed. 1997. *Approximation Algorithms for NP-Hard Problem*. PWS Publishing Company.

HOLLANDER, Y. 1995. Search algorithms for cache memory. Ph.D. thesis, Technion – Israel Institute of Technology.

Intel 2001a. *Desktop Performance and Optimization for Intel$^{(R)}$ Pentium$^{(R)}$ 4 Processor*. Intel. `http://developer.intel. com/design/pentium4/papers/249438.htm`.

Intel 2001b. *Intel$^{(R)}$ Pentium$^{(R)}$ 4 Processor Optimization, Reference Manual*. Intel. `http://developer.intel.com/design/pentium4/manuals/248966.htm`.

KANN, V., KHANNA, S., LAGERGREN, J., AND PANCONESI, A. 1997. On the hardness of approximating max k-cut and its dual. *Chicago Journal of Theoretical Computer Science 1997,* 2 (June).

KENNEDY, K. AND KREMER, U. 1998. Automatic data layout for distributed-memory machines. *ACM Trans. Program. Lang. Syst. 20*, 869–916.

MCKINLEY, K. S., CARR, S., AND TSENG, C.-W. 1996. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst. 18,* 4, 424–453.

PETRANK, E. AND RAWITZ, D. 2002. The hardness of cache conscious data placement. In *29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 101–112.

RIVERA, G. AND TSENG, C.-W. 1998. Eliminating conflict misses for high performance architectures. In *International Conference on Supercomputing*. 353–360.

SCHMIDT, W. J., ROEDIGER, R. R., MESTAD, C. S., MENDELSON, B., SHAVIT-LOTTEM, I., AND BORTNIKOV-SITNITSKY, V. 1998. Profile-directed restructuring of operating system code. *IBM Systems Journal 37,* 2, 270–305.

THABIT, K. O. 1982. Cache management by the compiler. Ph.D. thesis, Rice University.