On the Complexity of Direct Caching

Y. Hollander    A. Itai *
Computer Science Dept.
Technion, Haifa, Israel

January 27, 1994

Abstract

In this paper we study positioning strategies for improving the performance of a memory system with a direct mapped cache. A positioning technique determines for every program item, (instruction or data), its address in main memory.

We break the general positioning problem into two, the collision minimization and the grouping problems. We show that both the mapping and the grouping problems are NP-complete and that the mapping problem is also hard to approximate.

In addition we look at probabilistic models. While in the Independent Reference Model (IRM), the optimal strategies can be found by simple polynomial algorithms [2], in the more realistic Markov model we show that the grouping problem is NP-complete.

*Partially supported by the Foundation for Promotion of Research at the Technion
1 Introduction

Typical storage systems consist of a hierarchy of several levels: the topmost level is small, fast, and expensive; each successive level is larger, slower and cheaper. This structure has been promoted since it offers expected access time close to that of the fastest level while keeping the average cost per memory cell near the cost of the cheapest level. In this paper we concentrate on the top two levels of the hierarchy, i.e., the relatively small processor cache and the much larger main memory below it.

The term program item will be used for both instructions and data that the program reference in a single memory access. Both cache and main memory are organized as an array of pages with identical and fixed page size. To access an item, the processor first checks whether the memory page holding the item is in the cache. If it is not there, a cache miss occurs and the entire memory page is loaded into the cache. Due to the huge difference in the access times between cache and main memory, the performance penalty caused by a main memory access is significant. Therefore, memory system performance is measured by the hit ratio, \( h/a \), where \( h \) is the number of cache hits (accesses that are not cache misses) and \( a \) is the total number of memory accesses. Cache organizations that exhibit best hit ratios are typically large and require complex management.

Since processor speeds have increased much faster than memory speeds, performance has become increasingly dependent on yet another factor – the cache access time. The best access time is achieved by small caches with simple management schemes. To this end, mappings are required to be direct – each memory page is mapped to a single predetermined cache page that is determined only by the memory page. However, such caches have lower hit ratios. Extensive comparisons [4] have shown that in this tradeoff between high hit ratio and minimal access time, direct mapped caches are the preferred solution. Therefore, current research in the field has focused on improving the hit ratio of direct mapped caches by considering different mappings of the program and its data to the memory pages.

The number of cache misses is minimized in two steps: grouping the program items into groups, and minimizing the inter-group collisions. Let \( i \) and \( i' \) be two program items residing in main memory pages \( m, m' \), which are mapped to the same cache page \( c \). Whenever \( i' \) is referenced after \( i \) a cache miss occurs. These misses, called collisions, could be avoided by moving \( i' \) to another memory page \( m'' \) that is mapped to a cache page \( c' \neq c \).

This collision minimization problem can be defined formally: given the access sequence of the program, find an assignment of the program items to memory pages such that the number of misses is minimal. In Section 3 we show that this problem is NP-complete. Moreover, in Section 3.3 we show that it is also hard to approximate.

On the other hand, grouping techniques exploit the fact that whenever a main memory page is loaded into the cache, additional items (those sharing the same page)
are also loaded, and if the next access is to one of these items, no cache miss occurs. In particular, if all items mapped to the same main memory page are accessed consecutively, then only the first reference of each page is a miss. Therefore, it is desirable to group items in the same main memory page if they are typically accessed one after the other.

The grouping problem can be defined formally as the following: given the access sequence of the program and the page capacity \( b \), find a partitioning of the program items to groups of at most \( b \) such that the number of misses is minimal.

In Section 4 we show that for a cache with a single page that can hold at most two items this problem is equivalent to matching and, hence, solvable in polynomial time. For larger pages we show that the problem is NP-complete.

As a first approximation, Gal et al. [2] have used the Independent Reference Model (IRM) to model the program’s access pattern. In this model they assumed that the probability to access an item is known and is independent of previous references. For this model they derived optimal algorithms for both problems.

In contrast in Section 5 we show that if we consider the access probabilities as Markovian random variables, then finding the optimal grouping is NP-Complete.

# 2 The Model

We first define both the program access pattern representation and the cache model.

## 2.1 The Program Model

Programs reference both instructions and data elements, which we collectively call program items and number them \( \{1, 2, \ldots, n\} \). We shall assume that all program items have the same size.

The program’s runtime behavior can be represented in several ways, the most detailed of which is its trace. The trace is an ordered list of all memory references made by the program for a specific input.

Consider a memory system with \( k \) pages, \( s_0, \ldots, s_{k-1} \), of fast memory (the cache) and \( n > k \) pages, \( v_0, \ldots, v_{n-1} \), of slow memory. To satisfy a request the page must be in fast memory. If the requested page, \( v_i \), is not in cache, a page fault occurs. The page is loaded from memory to page \( s_j \) of the cache, overwriting the page previously residing there. In contrast to the paging problem [7], page \( v_i \) of memory can be loaded to only a single predetermined page of the cache \( s_j \). The mapping between memory pages and the cache pages is given by the function \( f: \{0, \ldots, n-1\} \rightarrow \{0, \ldots, k-1\} \).
The cache mapping problem

3.1 Problem definition

Let $\sigma$ be a sequence of requests that must be satisfied in their order of occurrence. $C_f(\sigma)$ denotes the number of page faults that occurred while servicing $\sigma$ using the mapping function $f$. Given $\sigma$, the cache mapping problem ($k$-CM) is the problem of finding a mapping function $f$ that minimizes $C_f(\sigma)$.

3.2 $k$-CM is NP-Complete

The $k$-CM problem is defined as follows:

**Input:** A sequence $\sigma$ of requests and an integer $m > 0$.

**Question:** Is there a mapping function $f$ to a $k$ page cache such that $C_f(\sigma) \leq m$.

First we present the result for $k = 2$.

**Theorem 1** $2$-CM is NP-Complete.

**Proof:** First, note that $2$-CM is NP since it is possible to guess $f$ and to calculate $C_f(\sigma)$. To show that $2$-CM is NP-Complete, we give a reduction from the Simple Max Cut problem (SMAXC)[3]. Given $G = (V, E)$ and $S \subseteq V$, the cut $(S; \bar{S})$ is the set of edges in $G$ such that one of their endpoints is in $S$ and the other is in $\bar{S} = V - S$. The SMAXC problem is defined as follows:

**Input:** A graph $G = (V, E)$ and a positive integer $m$.

**Question:** Is there a subset $S$ of vertices such that $|(S; \bar{S})| \geq m$.

Given $G = (V, E)$ where $V = \{v_1 \ldots v_n\}$ are the vertices of $G$ we construct the sequence of requests $\sigma_G$. The set of memory pages consists of $V \cup \{x, y\}$ where $x \neq y \notin V$. The sequence $\sigma_G$ starts with a separator string $xyxyxy$. Then, for every edge $e = (v_i, v_j) \in E$ we add the the string $v_i v_j v_i v_j$ followed by the separator string. Note that the construction of $\sigma_G$ takes polynomial time in $n$ since $|\sigma_G| = 10|E| + 6$.

**Claim 1** Let $f$ be an optimal mapping function for $\sigma_G$. Then, $f(x) \neq f(y)$.

**Proof:** Let $p(e) = v_i v_j v_i v_j xyxyxy$ for $e = (v_i, v_j) \in E$. If $f$ maps both $x$ and $y$ to the same cache page ($f(x) = f(y)$) then $C_f(p) \geq 7$ since the separator part causes at least six page faults and the prefix contribution is at least one. On the other hand, if $f(x) \neq f(y)$ then $C_f(p) \leq 6$ since the separator part contributes at most two page faults and the prefix part contributes at most four faults. □

The theorem follows from the following claim:
Claim 2 Let \( G \) be a graph and construct \( \sigma_G \) as above. Then \( G \) has a cut \((S, \bar{S})\) of size \( c \) if and only if there exists a mapping \( f : V \cup \{x, y\} \rightarrow \{0, 1\} \) of cost \( 2 + 5|E| - c \).

Proof: Given the cut \((S, \bar{S})\) construct \( f \) as follows:

\[
f(v) = \begin{cases} 
0 & \text{if } v \in S \cup \{x\} \\
1 & \text{if } v \in \bar{S} \cup \{y\}. 
\end{cases}
\]

Since \( f(x) \neq f(y) \) the initial subsequence \( xyxyxy \) costs two misses and after executing any subsequence \( xyxyxy \) the cache consists of \( \{x, y\} \). Let \( e = (u, v) \), and consider the cost of \( p(e) = uvwxyxyxy \).

Assume first that \( f(u) \neq f(v) \). Executing \( uvw \) costs two misses and executing \( xyxyxy \) costs an additional two misses. Hence, \( C_f(p(e)) = 4 \). When \( f(u) = f(v) \), then \( C_f(uvwxyxyxy) = 4 \) and \( C_f(xxyyxyyxy) = 1 \). Thus \( C_f(uvwxyxyxy) = 5 \).

\[
C_f(\sigma_G) = C_f(xxyyxyyxy) + \sum_{e \in E} C_f(p(e)) = 2 + \sum_{e \in (S, \bar{S})} C_f(p(e)) + \sum_{e \in (S, \bar{S})} C_f(p(e))
\]

\[
= 2 + 4|(S, \bar{S})| + 5(|E| - |(S, \bar{S})|) = 2 + 5|E| - |(S, \bar{S})|.
\]

For the other direction, let \( f \) be a mapping such that \( C_f(\sigma_G) = 2 + 5|E| - c \). Construct \( S = \{v : f(v) = 1\} \). As before,

\[
C_f(\sigma_G) = C_f(xxyyxyyxy) + 5|\{(u, v) \in E : f(u) \neq f(v)\}| + 4|\{(u, v) \in E : f(u) = f(v)\}|
\]

\[
= 2 + 5(|E| - |(S, \bar{S})|) + 4|(S, \bar{S})| = 2 + 5|E| - |(S, \bar{S})|.
\]

Hence, \(|(S, \bar{S})| = c \). \qed

Theorem 2 \( k\)-CM is \( NP\)-Complete.

Proof sketch: The proof is by reduction from 2-CM. Given \( \sigma = y_1y_2 \ldots y_i \) we use \( k-2 \) new additional slow memory pages \( x_1x_2 \ldots x_{k-2} \) to construct the new sequence of requests \( \sigma' = y_1x_1x_2 \ldots x_{k-2} \ldots y_ix_1x_2 \ldots x_{k-2} \) i.e., every request \( x_i \in \sigma \) is followed by the separator \( x_1x_2 \ldots x_{k-2} \).

Note that any optimal mapping function \( f \) assigns each \( x_i \) to a different cache page. Otherwise, every two pages \( x_i \) and \( x_j \) would contribute \( 2l \) to \( C_f(\sigma') \). Also, such \( f \) can avoid mapping any of the original memory pages to a cache page that an \( x_i \) is mapped to since it can only increase \( \sigma' \). Thus, the optimal \( f \) assigns the pages in \( \sigma \) to only two cache pages giving the optimal function for the case of the two pages cache. \qed

3.3 Approximating \( k\)-CM

MAX-SNP [6] is a class of optimization problems that can be approximated with constant error. It has been shown that several common optimization problems, e.g.,
MAX-SAT, are complete in MAX-SNP with respect to L-reductions which are transformations that preserve approximability.

Let II and II' be two optimization problems. II L-reduces to II' if there are polynomial time algorithms $F_1, F_2$, and constants $\alpha, \beta > 0$ such that for every instance $I$ of II:

1. $F_1$ produces an instance $I' = F_1(I)$ of II', such that the optimum of $I$ and $I'$, $OPT(I)$ and $OPT(I')$, respectively, satisfy $OPT(I') \leq \alpha OPT(I)$.
2. Given any solution of $I'$ with cost $c'$, algorithm $F_2$ produces a solution of $I$ with cost $c$ such that $|c - OPT(I)| \leq \beta |c' - OPT(I')|$.

In [1] the hardness of approximating MAX-SAT was shown by proving that there exists a constant $\epsilon > 0$ such that, unless $P = NP$, one cannot distinguish in polynomial time between formulae that all their clauses can be satisfied and formulae that only a fraction $1 - \epsilon$ of their clauses can be satisfied. This result gave a tool for dealing with the hardness of approximations. It implied that all complete problems in MAX-SNP cannot be approximated within any desired constant and hence, ruled out the possibility of finding a FPAS (Fully Polynomial Approximation Scheme) for these problems. We will show that the 2-CM problem is MAX-SNP-Hard.

**Theorem 3** 2-CM is MAX-SNP-Hard.

**Proof:** For showing that 2-CM is complete in NP we used a reduction from the MAX-CUT problem. Here we apply the same reduction on the MAX-CUT-B problem which is identical to the original problem except for the fact that the degree of every vertex in the graph is at most a constant $B$. Since MAX-CUT-B is complete in MAX-SNP [6] it is left to show that for the MAX-CUT-B problem the same reduction is an L-reduction.

Let $G$ be a graph with $|V| = n$ and maximal vertex degree $B$. Let $d^*$ denote size of the maximal cut in $G$. We first derive a lower bound on $d^*$ by using the fact that the MAX-2-COLORABILITY problem is isomorphic to the MAX-CUT problem. Consider the problem of finding a $k$-coloring of a given graph $G$ that maximizes the number of edges which are adjacent to two vertices of different colors.

**Definition 1** Consider a graph $G$ and a coloring of its vertices $\text{color} : \{1, 2 \ldots k\}$. An edge $e = (v_i, v_j)$ is consistent with respect to color, if $\text{color}(v_i) \neq \text{color}(v_j)$.

MAX-$k$-COLORABILITY is defined as follows:

**Input:** A graph $G = (V, E)$.

**Problem:** Find the maximum number of consistent edges in $G$ where the maximum is taken over all $k$-colorings of the vertices of $G$. 
It is easy to see that the MAX-CUT problem is isomorphic to the MAX-2-COLORABILITY problem. The cut size is exactly the number of consistent edges and hence $d^*$ is also the maximal number of consistent edges.

**Claim 3** For a connected graph $G$, $d^* \geq \frac{n}{B+1}$.

**Proof:** It is known that a graph $G$ with bounded degree $B$ can be colored by $B+1$ colors. Therefore, the size of the maximal color set, $MC$, (the set of vertices having the same color) is at least $\frac{n}{B+1}$. But since $G$ is connected and we have a valid coloring, every vertex $v \in MC$ has at least one edge $e = (v, u)$ such that $\text{color}(v) \neq \text{color}(u)$, i.e., $e$ is consistent. Thus, the number of consistent edges ($d^*$) is at least of size $\frac{n}{B+1}$.

□

Now we show that the two conditions for an L-reduction are satisfied.

**Claim 4** Let $G = (V, E)$ be a connected graph with $n > 2$ and let $F_1$ be the algorithm which given $G$ constructs $\sigma_G$. There exists a constant $\alpha$ such that $2 + 5|E| - d^* \leq \alpha d^*$.

**Proof:** Since the maximal degree in $G$ is bounded by $B$, $|E| \leq \frac{nB}{2}$. From Claim 3 it follows that: $n \leq d^* (B+1)$. Also, it is easy to see that for any graph with $n > 2$, $d^* \geq 2$. Thus,

\[
2 + 5|E| - d^* \leq 2 + 5nB - d^* \\
\leq 2 + 5B(B+1)d^* - d^* \\
\leq 5B(B+1).
\]

Therefore, taking $\alpha = 5B(B+1)$ satisfies the above inequality. □

**Claim 5** Let $c'$ be the solution cost of the 2-CM problem. There are a polynomial time algorithm $F_2$ and a constant $\beta$ such that $F_2$ produces a solution to the MAX-CUT-B problem such that its cost $c = 2 + 5|E| - c'$ and that $d^* - c \leq \beta (c' - (2 + 5|E| - d^*))$.

**Proof sketch:** Given a mapping function $F_1$ with cost $c'$ we know from Claim 2 that $F_2$ should take $S = \{v : f(v) = 1\}$ and that $|(S; \bar{S})| = c$. Then, choosing $\beta = 1$ satisfies the above inequality. □

### 4 The grouping problem

#### 4.1 Problem definition

In order to isolate the effect of increasing the page capacity has on the number of page faults, we assume a cache that has a single page and the page capacity, i.e, the number of items that fit into a page, is greater than one. In such a cache, the mapping function
is trivial and the only way to reduce the number of page faults is to take advantage of
the fact that when a fault occurs, not only the referenced item is loaded into the cache,
but also all other items that belong to the same page are loaded. Thus, accessing any
of one of the items in this page will not cause an additional page fault.

Again, we assume the requests must be satisfied in their order of occurrence. Let
\( I = \{0 \ldots n - 1\} \) be the set of the program items, \( k \) the capacity of a page, and
\( \sigma = x_1 \ldots x_m \) be a sequence of requests.

**Definition 2** A function \( g : I \rightarrow \{0 \ldots l - 1\} \) is a \( k \)-grouping function if \( |g^{-1}(j)| \leq k \)
for \( 0 \leq j \leq l - 1 \).

Let
\[
\chi^g_{x_i, x_j} = \begin{cases} 
1 & \text{if } g(x_i) \neq g(x_j) \\
0 & \text{otherwise.}
\end{cases}
\]

Then, the cost of the sequence \( \sigma \), given a \( k \)-grouping function \( g \) is,
\[
C_g(\sigma) = \sum_{i=2}^{l-1} \chi^g_{x_i, x_{i-1}}.
\]

The \( k \)-GROUP problem is defined as follows:

**Input:** A sequence \( \sigma \in I^* \) and an integer \( J \).

**Output:** Is there a \( k \)-grouping function \( g \) that maps \( I \) to \( \{0 \ldots l - 1\} \), such that
\( C_g(\sigma) \leq J \).

### 4.2 \( k \)-GROUP is NP-Complete

**Theorem 4** For \( k \geq 3 \) the \( k \)-GROUP problem is NP-Complete.

**Proof:** \( k \)-GROUP is in NP since it is possible to undeterministically choose a \( k \-
grouping function and then calculate its cost. To show that \( k \)-GROUP is NP-Complete,
we give a reduction from the \( k \)-GP problem (a variant of the GRAPH PARTITIONING
problem [3]). Let \( G = (V, E) \) be a graph with nonnegative weights \( w(e) \) for \( e \in E \).

**Definition 3** \( P = (V_1, V_2 \ldots V_l) \) is a \( k \)-partition of \( V \) if the sets \( V_j \) are disjoint and
\( |V_j| \leq k \) for \( 1 \leq j \leq l \).

**Definition 4** Let \( P \) be a \( k \)-partition of \( V \), then
\[
CR_P(G) = \{ e \in E \mid e \text{'s endpoints are in two different sets } V_j \}.
\]

The \( k \)-GP problem is the following:
Input: Graph $G = (V, E)$, an integer $J$.

Output: Is there a $k$-partition $P$ of $V$ such that $\sum_{e \in CR_P(G)} w(e) \leq J$.

This problem is known to be NP-Complete for $k \geq 3$ even for the case when $w(e) = 1$ for $e \in E$ [3].

Given $G = (V, E)$ with $V = \{v_1 \ldots v_n\}$ and an integer $J$, we construct the corresponding sequence of requests, $\sigma_G$. The set of possible requests, $I_G$, in $\sigma_G$ consists of $V \cup \{x_1 \neq x_2 \neq \ldots \neq x_k \notin V\}$. For every edge $e = (v_i, v_j) \in E$ we add to $\sigma_G$ the string $v_i v_j x_1 \ldots x_k x_1 \ldots x_k x_1 \ldots x_k$ (four repetitions of the separator string). This construction takes polynomial time in $|V|$ since $|\sigma_G| = (4k + 2)|E|$.

Claim 6 Let $G$ be a graph and construct $\sigma_G$ as above. Then there exists a $k$-partition $P$ of $G$ with $m = |CR_P(G)|$ if and only if there exists a $k$-grouping function $g : V \cup \{x_1 \ldots x_k\} \rightarrow \{0 \ldots l-1\}$ in which $g(x_1) = \ldots = g(x_k)$ with $C_g(\sigma_G) = 2|E| + m$.

Proof: Given $g$, let $P = (V_0 \ldots V_{l-1})$ where $V_i = \{v : g(v) = i\}$ for $0 \leq i \leq l-1$. Obviously, $P$ is a $k$-partition of $V$ since $g$ is a $k$-grouping. Also, note that for every edge $e = (u, v)$ the cost of the sequence $uv$ followed by four times the separator string is two if $g(u) = g(v)$ and three otherwise. Thus,

$$C_g(\sigma_G) = 3|\{e = (v, u) : g(u) \neq g(v)\}| + 2|\{e = (v, u) : g(u) = g(v)\}| + 2|E|$$

Hence, $|CR_P(G)| = m$.

On the other hand, given $P$ let $g(v) = l$ if $v \in V_i$ and $g(x) = l$ for $x \in I_G - V$. Note that for an edge $e = (u, v) \in E$ the cost of the sequence $C_g(uv) = 1$ if $e \notin CR_P(G)$ and two otherwise. Thus,

$$C_g(\sigma_G) = \sum_{e \in E} (1 + C_g(uv)) = |CR_P(G)| + 2|E|.$$

Claim 7 Let $g$ be an optimal grouping function for $\sigma_G$, then $g(x_1) = g(x_2) = \ldots = g(x_k)$.

Proof: If $g(x_i) \neq g(x_j)$ for some $i, j$, then $C_g(\sigma_G) \geq 4|E|$. Alternatively, if $g(x_1) = g(x_2) = \ldots = g(x_k)$ then $C_g(\sigma_G) \leq 3|E|$.

Combining the above two claims gives the result.
4.3 An optimal algorithm for k=2

For \( k = 2 \) the GRAPH PARTITIONING problem can be optimally solved by maximal weighted matching [3]. We show a polynomial reduction from 2-GROUP to 2-GP, thus giving an optimal algorithm for the 2-GROUP problem which runs in polynomial time.

**Definition 5** Let \( \sigma \in I^* \) be a request sequence. For \( i < j \in I \), let \( A\text{F}_{i,j}(\sigma) \) be the number of times \( i \) comes immediately after or before \( j \) in \( \sigma \).

**Theorem 5** There exists a polynomial algorithm for the 2-GROUP problem.

**Proof:** Given a sequence \( \sigma \), let \( G_\sigma = (V, E) \) be the graph in which \( V = I \) and an edge \( e = (i, j) \) with weight \( A\text{F}_{i,j}(\sigma) \) exists only if \( A\text{F}_{i,j}(\sigma) > 0 \). Since we may assume, w.l.o.g., that \( |I| \leq |\sigma| \), the above construction is linear in \( |\sigma| \). Now it is left to show that the optimal 2-partition on \( G_\sigma \) induces an optimal 2-grouping on \( \sigma \).

**Claim 8** Given a sequence \( \sigma \) construct the graph \( G_\sigma \) as above. There exists a 2-partition of \( G_\sigma \) such that \( \sum_{e \in CR_P(G_\sigma)} w(e) = m \) if and only if there exists a 2-grouping function \( f : I \to \{0, 1\} \) such that \( C_f(\sigma) = m \).

**Proof:** Given \( P \), the 2-grouping function \( f \) is defined to be \( f(i) = j \) if \( i \in V_j \). Obviously, \( f \) is a 2-grouping since \( P \) is a 2-partition. In the other direction, given \( f \) define a 2-partition \( P = (V_0 \ldots V_{l-1}) \) such that \( V_i = \{v : f(v) = i\} \). Again, it is easy to see that \( P \) is a 2-partition. Now it is left to verify that \( C_f(\sigma) = \sum_{e \in CR_P(G_\sigma)} w(e) \).

\[
C_f(\sigma) = \sum_{i=2}^{|I|} \chi\text{f}_{\sigma_i,\sigma_{i-1}}
= \sum_{i<j, f(i) \neq f(j)} A\text{F}_{i,j}
= \sum_{i<j, i \in V_a, j \in V_b, a \neq b} A\text{F}_{i,j}
= \sum_{e \in CR_P(G_\sigma)} w(e)
\]

5 The Markov Model

In previous sections the program’s runtime is represented by its trace. Since usually traces are huge, considering the full trace is often impractical. One alternative is to use a sample of the trace and hope that it accurately represents the entire trace.
Since the behavior should depend on the entire program, we can take a probabilistic approach. As a first approximation Gal et al. [2] assumed that the references are independent, i.e., the program trace is modeled as a sequence of independent, identically distributed random variables $Y_t$ with:

$$Pr(Y_t = j) = p_j \text{ for all } t \text{ and } 1 \leq j \leq n.$$  

Thus, the runtime behavior of a program with $n$ items is represented by appearance probabilities $p_1, \ldots, p_n$. This model ignores locality of reference. Algorithms, such as LRU, which exploit this locality are known to be very successful in practice.

Another alternative is to assume that the current memory reference depends only on the previous one. Then, the model for the program trace is a Markov chain. For any two items $i, j$, $p_{i,j}$ is the conditional transition probability, i.e., the probability that the next accessed item is $j$ given that the last element accessed was $i$. These probabilities can be easily estimated by processing the trace.

## 5.1 The problem definition

Let $I = \{0\ldots n-1\}$ be the set of the program items, $p_{i,j}$ the probability that item $j$ is referenced immediately after item $i$ was referenced, and $k$ the capacity of a page. We assume the requests are generated by an ergodic Markov chain $M$ with $I$ the set of states and $P = (p_{i,j})$ the transition probabilities.

Let $\pi(i)$ denote the probability that $M$ is at state $i$ at steady state, and let $g$ be a $k$-grouping function. Given $M$ and $g$ the cache hit probability is:

$$Pr_{M,g}(hit) = \sum_{i \in I} \pi(i) \sum_{k \in g^{-1}(i)} p_{i,k}.$$  \hfill (1)

The $k$-MrGROUP problem is the follows:

**Input:** An ergodic Markov chain $M$ and a probability $h$.

**Question:** Is there a $k$-grouping function $g$ such that $Pr_{M,g}(hit) \geq h$?

**Theorem 6** The 3-MrGROUP problem is NP-Complete.

**Proof:** The 3-MrGROUP problem is in NP since it is possible to nondeterministically choose a $k$-grouping function, compute the steady state probability distribution [5], and compute the cache hit probability. To show that 3-MrGROUP is NP-Complete, we give a reduction from the partition to triangles (PIT) problem [3]. The PIT problem is defined as follows:

**Input:** A Graph $G = (V, E)$, with $|V| = 3q$ for a positive integer $q$. 


**Question:** Is there a partition of $V$ into $q$ disjoint sets $V_1 \ldots V_q$ of three vertices each such that, for each $V_i = \{v_1, v_2, v_3\}$ the three edges $(v_1, v_2), (v_1, v_3)$ and $(v_2, v_3)$ all belong to $E$?

Given $G = (V, E)$ we construct the corresponding Markov chain $M_G = (p_{i,j})$. The set of states is $V \cup \{A, B, C\}$ where $A, B, C \notin V$. The transition probabilities:

$$\begin{align*}
p_{i,j} &= \begin{cases} 
\frac{1}{n} & (i, j) \in E \\
1 - \frac{\text{deg}(i)}{n} & i \in V \text{ and } j = A \\
1 & i = A \text{ and } j = B \\
1 & i = B \text{ and } j = C \\
\frac{3}{4} & i = C \text{ and } j = A \\
\frac{1}{4n} & i = C \text{ and } j \in V
\end{cases} \quad (2)
\end{align*}$$

**Claim 9** If $G$ is a connected graph, then $M_G$ is ergodic.

**Proof sketch:** Since $G$ is connected it is possible in a finite number of steps with a positive probability to reach $A$ from every state $v \in V$, reach $C$ from $A$ and then every state $u \in V$ from $C$. Also, since both cycles $(A, B, C)$ and $(A, B, C, v_1)$ belong to $G$, $M_G$ is aperiodic.

**Lemma 1** Let $G$ be a connected graph and construct $M_G$ as above. Let $g$ be an optimal 3-grouping function for $M_G$. Then, $\Pr_{M,G} = \frac{1}{4} \pi(A) + \frac{2}{n} \sum_{i \in V} \pi(i)$ if and only if $G$ can be partitioned into triangles.

**Proof:**

**Claim 10** For $M_G$ the following equations hold:

$$\pi(A) = \pi(B) = \pi(C). \quad (3)$$

$$\pi(A) = \frac{3}{4} \pi(C) + \sum_{i \in V} \pi(i) \left(1 - \frac{d(i)}{n}\right) \quad (4)$$

which also implies that

$$\pi(A) = 4 \sum_{i \in V} \pi(i) \left(1 - \frac{d(i)}{n}\right). \quad (5)$$

**Proof sketch:** Elementary Markov chain theory.

**Claim 11** Let $G$ be a connected graph. There exists an optimal 3-grouping function $g$ on $M_G$ such that $g(A) = g(B) = g(C)$. 

11
Figure 1: The $M_G$ chain.
Proof: For $v \in V \cup \{A, B, C\}$, let $\text{group}(v) = \{u \mid g(u) = g(v)\}$ the items that are grouped at the same group. Let us assume that either $B \notin \text{group}(A)$ or $C \notin \text{group}(A)$. We will show how to modify $g$ to $g'$ so that $g'(B) = g'(C) = g'(A)$ without decreasing the hit probability, i.e., $\Delta = \Pr_{M_i}(\text{hit}) - \Pr_{M_i}(\text{hit}) \geq 0$.

First we show that putting $A$ and $B$ in the same group does not decrease the cache hit probability. We consider several possible cases: If $\left|\text{group}(A)\right| < 3$, i.e., $A$’s group is not full, then $B$ is moved to $\text{group}(A)$. If $g(C) \neq g(B)$ then $\Delta = \pi(A)$. Alternatively, if $g(C) = g(B)$ then $\Delta = 0$ since $\pi(A) = \pi(B)$.

If $\left|\text{group}(A)\right| = 3$, i.e., $A$’s group is full and an exchange is made. Let $x \in \text{group}(A)$ such that $x \neq A, C$. There are several subcases to consider.

1. $g(B) \neq g(C)$: In this case $x$ is exchanged with $B$. If $C \notin \text{group}(A)$ then there exists $y \in \text{group}(A), y \neq A, C$ and:

$$\Delta \geq \pi(A) - \pi(x) \left(1 - \frac{d(x)}{n}\right) - \frac{\pi(x)}{n} - \frac{\pi(y)}{n}.$$ 

Since $1 - \frac{d(x)}{n} \geq \frac{1}{n}$ for each $v \in V$, equation 5 implies that

$$\pi(x) \left(1 - \frac{d(x)}{n}\right) + \frac{\pi(x)}{n} + \frac{\pi(y)}{n} \leq 2 \sum_{i \in V} \pi(i) \left(1 - \frac{d(i)}{n}\right) \leq \pi(A)$$

and therefore the change in the hit probability is non-negative.

If, however, $C \in \text{group}(A)$ then:

$$\Delta \geq \pi(A) + \pi(B) - \pi(x) \left(1 - \frac{d(x)}{n}\right) - \pi(C) \frac{1}{4n}$$

which is non-negative (from equations 3 and 5).

2. $g(B) = g(C)$: In this case $A$ is either moved to $\text{group}(B)$ (if $\left|\text{group}(B)\right| = 2$) or exchanged with $x \in \text{group}(B), x \neq B, C$ (if $\left|\text{group}(B)\right| = 3$). In both cases the change in the hit ratio:

$$\Delta \geq \pi(A) + \frac{3}{4}\pi(C) - \frac{1}{4n}\pi(C) - \sum_{i \in \text{group}(A) - A} \pi(i) \left(1 - \frac{d(i)}{n}\right)$$

is non-negative (from equations 4 and 5).

Finally, assuming that $g(A) = g(B)$ we either move or exchange $C$ with $x$. In both cases it is easy to see that:

$$\Delta \geq \pi(A) + \frac{3}{4}\pi(C) - \sum_{i \in \text{group}(C) - C} \frac{1}{4n}\pi(C) - \pi(x) \left(1 - \frac{d(x)}{n}\right) \geq 0.$$
Now we can return to the proof of Lemma 1. First note that both the construction of $M_G$ and the computation of the hit probability take polynomial time in $V$.

By the above claim we may assume w.l.o.g. that $g(A) = g(B) = g(C)$. Thus, the contribution of the states $A, B, C$ to the cache hit probability for an optimal $k$-grouping function is:

$$
\pi(A) \times p_{A,B} + \pi(B) \times p_{B,C} + \pi(C) \times p_{C,A} = \frac{11}{4} \pi(A)
$$

If $G$ can be partitioned into triangles then every vertex $v \in V$ contributes $\frac{2}{n} \pi(v)$ to the hit probability since two of its neighbors are in the same triangle and therefore $Pr_{M,g} = \frac{11}{4} \pi(A) + \frac{2}{n} \sum_{i \in V} \pi(i)$. If $G$ cannot be partitioned into triangles then at least one vertex $u \in V$ contributes at most $\frac{1}{n} \pi(u)$ and $Pr_{M,g} < \frac{11}{4} \pi(A) + \frac{2}{n} \sum_{i \in V} \pi(i)$. □

References


