# Efficient Scheduling of Page Access in Join Processing

C.Y. Chan and B.C. Ooi
Dept. of Information Systems & Computer Science
National University of Singapore
Kent Ridge Crescent
Singapore 119260
Email: {chancy,ooibc}@iscs.nus.sg

**Abstract**

This paper examines the issue of scheduling page access in join processing. Two related problems are addressed: (1) the determination of an optimal page access sequence such that the join can be computed without any page reaccesses using the minimum number of buffer pages, and (2) the determination of an optimal page access sequence such that the join can be computed with the minimum number of page reaccesses given a limited number of buffer pages. The first problem is believed to be NP-hard while the second problem has been shown to be NP-complete. Efficient heuristics for both problems can optimize buffer utilization as well as disk I/O cost. By modeling a page access sequence as a concatenation of segments, we derived two desirable properties of a page access sequence, which serve as the basis of our new heuristics for solving both problems. An experimental performance comparison of the new heuristics with existing heuristics show that the new heuristics perform better than existing heuristics for the first problem, and also perform better for the second problem provided that the number of available buffer pages is not much less than the optimal buffer size.

**Keywords:** Heuristics, join graph, join processing, page access scheduling, page access sequence.

## 1 Introduction

The join operation is one of the most performance-critical operations in database systems due to its frequent execution and costly evaluation. In relational database systems, value-based joins are used to combine information from more than one relation (as a result of normalization). In object-oriented database systems, besides supporting the conventional value-based joins, identity-based joins are also needed to optimize the evaluation of path expressions [6] (or nested predicates [1]) to retrieve objects related by their class-composition hierarchy. Much work has been done in optimizing join operation [8, 4], and many join processing techniques are common to both relational and object-oriented database systems.

One strategy to compute the join $R_1 \bowtie R_2$ is the following 2-step **scan-join** algorithm:

(1) Scan the relevant index(es) on the two join relations to obtain a list of pairs of data page identifiers $(p, q)$ where page $p$ contains some tuple which needs to join with some other tuple in page $q$.

(2) Use the page-pair information from step (1) to schedule an efficient page access sequence to fetch the data pages to compute the join results.

The scan-join algorithm is applicable in many join processing strategies using indexes. For example, consider the join $R \bowtie S$ where each of the join relations has an unclustered $B^+$-tree index on the join attribute(s). Using the scan-join algorithm, the index leaf pages of both relations are sequentially scanned and merged to obtain a list of pairs of TIDs (tuple identifiers) of related tuples, from which a list of pairs of page identifiers can be derived, and the join computed by fetching the participating data pages. The scan-join algorithm has been shown to be more efficient than the hybrid-hash join algorithm when the join selectivity is low [3]. Unclustered indexes are common since there can be at most one clustered index for each relation. Furthermore, multi-dimensional indexes are generally unclustered. In fact, the scan-join algorithm is applicable in any join strategy that utilizes precomputed join results (in the form of a list of pairs of tuple identifiers or object identifiers) organized as index structures to expedite join computation in relational and/or object-oriented database systems. Many such indexes have been proposed: the identity index [6], the join index [12], the composite $B^+$-tree [2], the nested join index [1], and the access support relation [5]. All these indexes materialized the join results to expedite join processing.

The performance of the scan-join algorithm depends on two factors: the efficiency of the join index(es) (step 1) and the efficiency of the page access sequence schedule (step 2). While much research has been done in designing new join indexes, there is very little work on the issue of efficient page access sequence to fetch data pages, which we will address in this paper. An efficient page access sequence is important to the performance of the algorithm as it can impact both the utilization of the buffer as well as the number of disk I/O accesses (in terms of number of page reaccesses). Clearly, the number of page reaccesses is likely to increase with a smaller number of buffer pages allocated. This raises two interesting problems:

(1) What is the optimal page access sequence such that the join can be computed without any page reaccesses using the minimum number of buffer pages?

(2) Given a fixed number of buffer pages, what is the optimal page access sequence such that the join can be computed with the minimum number of page reaccesses?

We shall refer to both these problems as OPAS1 and OPAS2 respectively (for *optimal page access sequence problem*).

A solution to the OPAS1 problem provides not only the optimal page access sequence but also the optimal buffer size. This information can be used to allocate an optimal number of buffer pages to compute the join without incurring any unnecessary disk I/O, resulting in an efficient join computation in terms of buffer utilization as well as disk I/O. Furthermore, with the page access schedule, the optimizer can utilize this knowledge to allocate the buffer pages dynamically, resulting in better buffer utilization. This is possible since it is unlikely that the optimal number of buffer pages is all needed for the entire page access sequence, so buffer pages can be requested and released as necessary. On the other hand, when the optimal number of buffer pages is not available, a solution to the OPAS2 problem can be useful in two ways. Firstly, the optimal page access sequence for a limited buffer space provides an I/O efficient strategy to compute the join with the minimum number of page reaccesses. Secondly, as a solution to the OPAS2 problem also provides information on the number of page reaccesses incurred, the database system can make use of this information to either allocate a fraction or all of the available buffer pages to perform the join, or defer the join operation until more or the optimal number of pages is available, by considering the marginal gains of waiting for more buffer pages to be made available [9].

The OPAS1 problem, determining the least upper bound on the size of the buffer, is believed to be NP-hard [3, 10], while the OPAS2 problem, determining the least upper bound on the number of page reaccesses, has been shown to be NP-complete for the case when the buffer size is equal to two pages [7]. Efficient heuristics for these problems are important as they will optimize both buffer utilization and disk I/O. In this paper, we propose new heuristics for both the OPAS1 and OPAS2 problems. The results of performance experiments show that the new heuristics outperform existing heuristics for a wide range of cases, in particular, for different types and sizes of join graphs, and their edge ratios. The rest of this paper is organized as follows. In Section 2, we present some desirable properties of a page access sequence which serve as the basis for our new OPAS1 heuristic presented in Section 3. Section 4 surveys existing OPAS1 heuristics and compares them with our proposed heuristic qualitatively. Section 5 presents performance comparisons of the new heuristic with existing OPAS1 heuristics. In Section 6, we examine existing heuristics for OPAS2 problem which are essentially extensions of OPAS1 heuristics. We also propose a new OPAS2 heuristic by extending our new OPAS1 heuristic. Section 7 compares the performance of the new OPAS2 heuristic with existing OPAS2 heuristics. Finally we conclude in Section 8.

# 2 Desirable Properties of a Page Access Sequence

In this section, we derive two desirable properties of a page access sequence that minimize the number of buffer pages necessary to compute the join without any page reaccesses. We first define the concept of a segment and model a page access sequence as a concatenation of a finite number of segment. Based on this model, we derive two desirable properties of a page access sequence, namely that it should not contain premature page accesses and that it should be a minimal page access sequence. We then show that any page access sequence $S$ can be refined into another page access sequence $S'$ such that $S'$ satisfies both properties and $S'$ requires no more buffer pages to join than $S$.

The page-pair information of a join $R_1 \bowtie R_2$ can be represented by an undirected graph $G = (V, E)$ where $V$ represents the set of pages participating in the join, and $E \subseteq V \times V$ represents the set of page-pairs to be joined such that $(v_1, v_2) \in E$ iff there exists some tuple in page $v_1$ that joins with some tuple in page $v_2$. We shall refer to such a graph as a **join graph**. Other terminology that has been used to refer to such a graph includes page-pair graph [10] and page-connectivity graph [11, 3]. The join graph is bipartite if no page in $V$ contains tuples from both relations $R_1$ and $R_2$; i.e., the relations are not inter-clustered. For a bipartite join graph, we shall use $V_1$ and $V_2$ to denote the subset of pages in $V$ that belongs to $R_1$ and $R_2$, respectively.

Note that a join graph may consist of a number of connected components; however, since each connected component can be treated as a separate independent join computation, we shall for the rest of this paper, consider only connected join graphs.

**Definition 1** *Let $G = (V, E)$ be a join graph. A* **page access sequence** $S = < p_1, p_2, \ldots, p_{|V|} >$ *is a sequence of distinct page accesses from $V$ where $p_i$ denote the $i^{th}$ page fetched into the buffer.*

A page access sequence specifies the order of fetching the pages from the join graph into the buffer. Each newly fetched page is joined with the existing resident buffer pages and any page that has joined with all its adjacent pages in the join graph can be made available for subsequent page fetches.

**Definition 2** *Let $G = (V, E)$ be a join graph. The* **join set of a page** $p \in V$*, denoted by $J(p)$, is defined to be the set of all its adjacent pages in the join graph including $p$ itself; i.e., $J(p) = \{q \in V | (p, q) \in E\} \cup \{p\}$.*

Once all the pages in $J(p)$ have been fetched into the buffer, the buffer page occupied by $p$ can be made available for subsequent page fetches.

Each page access sequence $S$ is associated with two unique sequences of set of pages, $< B_1, B_2, \ldots, B_{|V|} >$ and $< D_1, D_2, \ldots, D_{|V|} >$, where $B_i$ denote the set of resident buffer pages after page $p_i$ has been fetched into the buffer and $D_i$ denote the set of buffer pages that are made available after page $p_i$ has been fetched and joined with the other resident buffer pages. The following equations characterize the relationship between $B_i$ and $D_i$:

$$B_0 = D_0 = \emptyset \quad (1)$$

$$B_i = B_{i-1} - D_{i-1} \cup \{p_i\} \quad \text{for } i = 1, 2, \ldots, |V| \quad (2)$$

$$D_i = \{p \in B_i | J(p) \subseteq B_i\} \quad \text{for } i = 1, 2, \ldots, |V| \quad (3)$$

Since $p_{|V|}$ is the last page fetched into the buffer, we must have $D_{|V|} = B_{|V|}$.

**Definition 3** *Let* $S = < p_1, p_2, \ldots, p_{|V|} >$ *be a page access sequence. The* **upper bound of S**, *denoted by* $MAX(S)$, *is defined to be the upper bound on the buffer size of* $S$; *i.e.,* $MAX(S) = \max\{|B_1|, |B_2|, \ldots, |B_{|V|}|\}$. $S$ *is an* **optimal page access sequence** *iff* $MAX(S) \leq MAX(S')$ *for all page access sequence* $S'$.

A join graph may have a number of distinct optimal page access sequences. Note that for the join to be computed without any page reacesses, $MAX(S) \geq \max\{|J(p)| : p \in V\}$. Furthermore, if $S$ is an optimal page access sequence for a bipartite join graph, then we must have $MAX(S) \leq \min\{|V_1|, |V_2|\} + 1$. This bound can be achieved using the nested-loop join algorithm.

**Definition 4** *Let* $S = < p_1, p_2, \ldots, p_{|V|} >$ *be a page access sequence.* $S' = < p_i, p_{i+1}, \ldots, p_{i+k} >$, $1 \leq i \leq |V| - k$, *is said to be a* **segment of a page access sequence** $S$ *iff* $S'$ *is a non-empty subsequence of* $S$ *such that (1)* $D_j = \emptyset$ *for* $i \leq j < i + k$, *(2)* $D_{i+k} \neq \emptyset$, *and (3)* $D_{i-1} \neq \emptyset$ *if* $i > 1$.

While the sequence of pages in a segment are fetched, no pages can be made available until the last page in the segment has been fetched. Each page access sequence $S$ can be uniquely expressed as a concatenation of a finite number of segments. We shall denote each segment of a page access sequence $S$ by $S_{j_i}$ where $j_i$ is the index of the last page in the segment; i.e., $S_{j_i} = < p_{j_{i-1}+1}, p_{j_{i-1}+2}, \ldots, p_{j_i} >$ with $j_0 = 0$, $i \geq 1$. So if $S$ is a page access sequence with $m$ segments, then

$$\begin{aligned} S &= S_{j_1} \circ S_{j_2} \circ \ldots \circ S_{j_m} \\ &= < p_1, \ldots, p_{j_1} > \circ < p_{j_1+1}, \ldots, p_{j_2} > \circ \ldots \circ < p_{j_{m-1}+1}, \ldots, p_{j_m} > . \end{aligned}$$

Since $D_{|V|} = B_{|V|} \neq \emptyset$, we must have $j_m = |V|$. We shall also use the same notation $S_{j_k}$ to refer to the set of pages contained in the segment $S_{j_k}$ when the context makes the intended meaning clear. Note that the last page in a segment $S_{j_i}$ must be contained in the join set of every page $d \in D_{j_i}$; otherwise some page would have been made available before the last page from the segment is fetched.

**Proposition 1** *Let* $S$ *be a page access sequence with* $m$ *segments. Then* $MAX(S) = \max\{|B_{j_1}|, |B_{j_2}|, \ldots, |B_{j_m}|\}$.

**Proof**: Consider the $k^{th}$ segment of a page access sequence $S$. By Definition 4, $S_{j_k} = < p_{j_{k-1}+1}, p_{j_{k-1}+2}, \ldots, p_{j_k} >$ such that $D_i = \emptyset$ for $j_{k-1} < i < j_k$. Therefore by Equation 2, we must have $|B_{i+1}| > |B_i|$ for $j_{k-1} < i < j_k$ which implies that $|B_{j_k}| = \max\{|B_i| : j_{k-1} < i \leq j_k\}$. Hence $MAX(S) = \max\{|B_1|, |B_2|, \ldots, |B_{|V|}|\}$ $= \max\{|B_{j_1}|, |B_{j_2}|, \ldots, |B_{j_m}|\}$. $\square$

**Definition 5** *A page* $p$ *in a segment* $S_{j_i}$ *is said to be a* **premature page access** *in* $S_{j_i}$ *iff* $\forall d \in D_{j_i}, p \notin J(d)$. $S$ *is said to be a* **premature page access sequence** *iff* $S$ *contains some premature page access(es).*

Informally, a page $p$ is a premature page access in a segment $S_{j_i}$ if $p$ is fetched within the segment but $p$ does not contribute to making any of the pages in $D_{j_i}$ available. This means that the fetching of $p$ can be postponed after segment $S_{j_i}$ without affecting $D_{j_i}$. Therefore to minimize the number of resident buffer pages and hence $MAX(S)$, premature page accesses should be eliminated.

**Proposition 2** *Let $S$ be a page access sequence with $w$ premature page accesses, $w \geq 1$. Then $S$ can be refined into another page access sequence $S'$ such that (1) $S'$ has $(w-1)$ premature page accesses, and (2) $MAX(S') \leq MAX(S)$.*

**Proof**: Let $S = S_{j_1} \circ S_{j_2} \circ \ldots \circ S_{j_m} = < p_1, p_2, \ldots, p_{|V|} >$ be a page access sequence such that $p_r$ is a premature page access in $S$. Suppose $p_r$ is contained in the $x^{th}$ segment of $S$; i.e., $j_{x-1} < r \leq j_x$. Since $p_r \in S_{j_x}$ is a premature page access, $p_r \notin D_{j_x}$. So $p_r \in D_{j_z}$ for some $x < z \leq m$. Let $S_{j_y}$ be the segment of $S$ such that:

(C1) $x < y \leq z$,

(C2) $\exists d' \in D_{j_y}$ such that $p_r \in J(d')$, and

(C3) $\forall x < i < y, \forall d \in D_{j_i}, p_r \notin J(d)$.

Define a new page access sequence $S'$ such that $S'$ is equivalent to $S$ except that page $p_r$ is scheduled to be fetched immediately before the first page in $S_{j_y}$ as depicted in Figure 1. Formally, let $S' = < p_{\pi(1)}, p_{\pi(2)}, \ldots, p_{\pi(|V|)} >$ be a permutation of $S$ where

$$\pi(i) = \begin{cases} i & \text{if } 1 \leq i < r, \\ i+1 & \text{if } r \leq i \leq j_{y-1} - 1, \\ r & \text{if } i = j_{y-1}, \\ i & \text{if } j_{y-1} + 1 \leq i \leq |V|. \end{cases}$$

Let $S' = S'_{k_1} \circ S'_{k_2} \circ \ldots \circ S'_{k_n}$. We claim that $n = m$; i.e., $S'$ has the same number of segments as $S$. By definition of $\pi$,

(P1) $< p_{\pi(1)}, \ldots, p_{\pi(j_{x-1})} > = < p_1, \ldots, p_{j_{x-1}} > = S_{j_1} \circ \ldots \circ S_{j_{x-1}}$. So $S'_{k_i} = S_{j_i}$ for $1 \leq i \leq x-1$ which implies that $B'_{k_i} = B_{j_i}$ and $D'_{k_i} = D_{j_i}$ for $1 \leq i \leq x-1$. Clearly, each $S'_{k_i}$ has the same number of premature page accesses as $S_{j_i}$ for $1 \leq i \leq x-1$.

(P2) $< p_{\pi(j_{x-1}+1)}, \ldots, p_{\pi(r-1)}, p_{\pi(r)}, \ldots, p_{\pi(j_x-1)} > = < p_{j_{x-1}+1}, \ldots, p_{r-1}, p_{r+1}, \ldots, p_{j_x} > = S_{j_x}$ with $p_r$ removed. So $S'_{k_x} = S_{j_x}$ without $p_r$. Thus $B'_{k_x} = B_{j_x} - \{p_r\}$. Since $p_r \notin D_{j_x}$, therefore $D'_{k_x} = D_{j_x}$. Thus $S'_{k_x}$ is a minimal segment. Since $p_r$ is a premature page access in $S_{j_x}$ but $p_r \notin S'_{k_x}$, therefore $S'_{k_x}$ has one less premature page access compared to $S_{j_x}$.

(P3) $< p_{\pi(j_x)}, \ldots, p_{\pi(j_{y-1}-1)} > = < p_{j_x+1}, \ldots, p_{j_{y-1}} > = S_{j_{x+1}} \circ \ldots \circ S_{j_{y-1}}$. So $S'_{k_i} = S_{j_i}$ for $x+1 \leq i < y$. By (C3) and the fact that $B'_{k_x} = B_{j_x} - \{p_r\}$, we must have $B'_{k_i} = B_{j_i} - \{p_r\}$ and $D'_{k_i} = D_{j_i}$, for $x+1 \leq i < y$. Clearly, each $S'_{k_i}$ has the same number of premature page accesses as $S_{j_i}$, for $x+1 \leq i < y$.

(P4) $< p_{\pi(j_{y-1})}, p_{\pi(j_{y-1}+1)}, \ldots, p_{\pi(|V|)} > = < p_{j_r}, p_{j_{y-1}+1}, \ldots, p_{|V|} > = < p_r > \circ S_{j_y} \circ \ldots \circ S_{j_m}$. So $S'_{k_y} = < p_r > \circ S_{j_y}$ with $B'_{k_y} = B_{j_y}$ and $D'_{k_y} = D_{j_y}$ since $p_r \in B_{j_y}$ and $p_r \in D_{j_y}$. Thus $S'_{k_y}$ is a minimal segment. $S'_{k_i} = S_{j_i}$ so that $B'_{k_i} = B_{j_i}$ and $D'_{k_i} = D_{j_i}$, for $y < i \leq m$. Clearly, each $S'_{k_i}$ has the same number of premature page accesses as $S_{j_i}$ for $y \leq i \leq m$.

Hence $S' = S'_{k_1} \circ S'_{k_2} \circ \ldots \circ S'_{k_m}$ with one less premature page access compared to $S$.

By (P1) to (P4), we have $|B'_{k_i}| = |B_{j_i}|$ for $1 \leq i < x$ and for $y \leq i \leq m$; and $|B'_{k_i}| = |B_{j_i}| - 1$ for $x \leq i < y$. Hence $MAX(S') \leq MAX(S)$. In particular, if $|B_{j_i}| = MAX(S)$ iff $x \leq i < y$, then $MAX(S') = MAX(S) - 1$. □

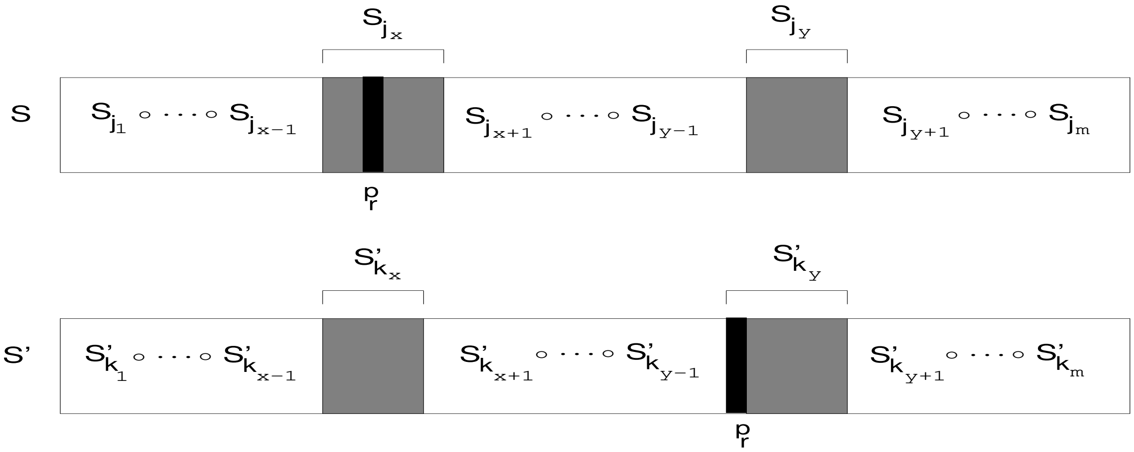



Figure 1: Refinement of $S$ into $S'$ to eliminate premature page access $p_r$

**Definition 6** *A segment $S_{j_k}$ is said to be a* **minimal segment** *iff $\forall d \in D_{j_k}$, $S_{j_k} \subseteq J(d)$. A page access sequence $S$ is a* **minimal page access sequence** *iff all its segments are minimal.*

Clearly, by Definitions 5 and 6, a minimal page access sequence must necessarily be a non-premature page access sequence. Intuitively, if a segment is minimal, then the sequence of fetching the pages in the segment is inmaterial in that it will not affect $MAX(S)$.

**Proposition 3** *If $S_{j_i}$ is a minimal segment of a page access sequence $S$, then every permutation of $S_{j_i}$ is also a minimal segment of $S$.*

**Proof**: Let $S_{j_i} = < p_{j_{i-1}+1}, \ldots, p_{j_i} >$ be a minimal segment of $S$ and $S' = < p'_{j_{i-1}+1}, \ldots, p'_{j_i} >$ be a permutation of $S_{j_i}$. Consider $D'_k$, $j_{i-1} + 1 \leq k < j_i$. Suppose $D'_k \neq \emptyset$; i.e., $D'_k$ contains some page $d$. Since $S'$ is a permutation of $S_{j_i}$, $D_{j_i} = \bigcup_{j_{i-1} < k \leq j_i} D'_k$. So $d \in D_{j_i}$. Note that $d \in D'_k$ implies that $p'_{k+1} \notin J(d)$. Since $p'_{k+1} \in S_{j_i}$, therefore $S_{j_i} \not\subseteq J(d)$ which contradicts the fact that $S_{j_i}$ is a minimal segment of $S$. So we must have $D'_k = \emptyset$ for $j_{i-1}+1 \le k < j_i$. Since $D_{j_i} = \bigcup_{j_{i-1} < k \le j_i} D'_k$, therefore $D'_{j_i} = D_{j_i} \ne \emptyset$. Hence $S'$ is a segment of $S$.

Suppose $S'$ is not minimal. By Definition 6, this implies that $\exists d \in D'_{j_i}, S' \not\subseteq J(d)$. But since $D'_{j_i} = D_{j_i}$, and both $S_{j_i}$ and $S'$ contain the same set of pages, this implies that $\exists d \in D_{j_i}$, $S_{j_i} \not\subseteq J(d)$ which contradicts the fact that $S_{j_i}$ is a minimal segment of $S$. Hence, $S'$ must be a minimal segment of $S$. □

**Proposition 4** *Let $S = S_{j_1} \circ S_{j_2} \circ \ldots \circ S_{j_m}$ be a page access sequence. Then $B_{j_i} = B_{j_{i-k}} \cup \bigcup_{i-k+1 \le t \le i} S_{j_t} - \bigcup_{i-k \le t \le i-1} D_{j_t}$, for $1 \le k < i$, $1 < i \le m$.*

**Proof**: From Equation 2, it can be established by induction that

$$B_i = B_{i-k} \cup \{p_t | i-k+1 \le t \le i\} - \bigcup_{i-k \le t \le i-1} D_t \text{ for } 1 \le k < i \le |V|.$$

So $B_{j_i} = B_{j_{i-k}} \cup \{p_t | j_{i-k}+1 \le t \le j_i\} - \bigcup_{j_{i-k} \le t \le j_i - 1} D_t$. Since $\forall S_{j_i}$ of $S$, $S_{j_i} = \{p_t | j_{i-1}+1 \le t \le j_i\}$ and $D_t = \emptyset$ for $j_i < t < j_{i+1}$, therefore $B_{j_i} = B_{j_{i-k}} \cup \bigcup_{i-k+1 \le t \le i} S_{j_t} - \bigcup_{i-k \le t \le i-1} D_{j_t}$, for $1 \le k < i \le m$. □

Informally, a page segment $S_{j_i}$ is minimal if every page fetched within the segment contributes to making every page in $D_{j_i}$ available. It follows that in a non-minimal segment $S_{j_i}$, there must be some page $d \in D_{j_i}$ which does not require all the pages in the segment to be fetched in order for it to be made available. Hence, by re-sequencing the page accesses in a non-minimal segment, the segment can be split into a number of shorter and minimal segments such that the number of resident buffer pages is reduced, which can potentially result in a smaller $MAX(S)$.

**Proposition 5** *Let $S$ be a non-premature page access sequence with $w$ non-minimal segments, $w \ge 1$. Then $S$ can be refined into another page access sequence $S'$ such that (1) $S'$ has $(w-1)$ non-minimal segments, and (2) $MAX(S') \le MAX(S)$.*

**Proof**: Let $S$ be a non-premature page access sequence with $m$ segments such that $S_{j_r}$ is its first non-minimal segment. Define a new page access sequence $S'$ such that $S'$ is equivalent to $S$ except that $S_{j_r}$ is permutated as follows:

$$\begin{aligned} S' &= (S'_{k_1} \circ \ldots \circ S'_{k_{r-1}}) \circ (S'_{k_r} \circ \ldots \circ S'_{k_{r+n-1}}) \circ (S'_{k_{r+n}} \circ \ldots \circ S'_{k_{m+n-1}}) \\ &= (S_{j_1} \circ \ldots \circ S_{j_{r-1}}) \circ \mathit{Permutate}(S_{j_r}) \circ (S_{j_{r+1}} \circ \ldots \circ S_{j_m}) \end{aligned}$$

where
- $(C1)$ $S'_{k_i} = S_{j_i}$ for $1 \le i \le r-1$
- $(C2)$ $S'_{k_i} = S_{j_{i-n+1}}$ for $r+n \le i \le m+n-1$
- $(C3)$ $\mathit{Permutate}(S_{j_r})$ is a permutation of $S_{j_r}$ such that for $1 \le i \le n$,
  - $(a)$ $S'_{k_{r+i-1}}$ is a minimal segment of $S'$ and
  - $(b)$ $|B'_{k_{r+i-1}}| < |B_{j_r}|$

We claim that $S_{j_r}$ can be permutated to satisfy (C3) by algorithm MINIMIZE (shown in Figure 2) with $S_{j_r}$ as the input parameter. We shall establish the correctness of the MINIMIZE algorithm by proving the following claims: (P1) If $F_i \neq \emptyset$, then $\forall p \in F_i, \exists d \in A_i,\ p \in J(d)$; (P2) Algorithm MINIMIZE terminates; (P3) $S'_{k_{r+i-1}}$ is a minimal segment of $S$; and (P4) $|B'_{k_{r+i-1}}| < |B_{j_r}|$.

(P1) Since $S_{j_r}$ is a segment, therefore by line 1, $F_1 = S_{j_r} \neq \emptyset$ and $A_1 = D_{j_r} \neq \emptyset$. Since $S$ (and hence $S_{j_r}$) has no premature page accesses, therefore by Definition 5, $\forall p \in F_1, \exists d \in A_1, p \in J(d)$. So the claim holds for $i = 1$. Assume that claim is true for $i = k$, $k \geq 1$. Consider $F_{k+1}$. Suppose $F_{k+1} \neq \emptyset$; i.e., $F_{k+1}$ contains some page $p$. Since $F_{k+1} = F_k - T_k$ (line 5), therefore $p \in F_k$ which implies, by the induction hypothesis, that $\exists d \in A_k$ such that $p \in J(d)$. Since $p \in F_{k+1}$ and $p \in J(d)$ for some $d \in A_k$, therefore by line 6, we must have $d \in A_{k+1}$. Hence by induction, the claim is true.

(P2) By (P1) and line 3, if $F_i \neq \emptyset$, then $T_i \neq \emptyset$. This implies that $F_{i+1} \subset F_i$ (by line 5). So eventually $F_{n+1} = \emptyset$ for some $n \geq 1$ and the algorithm terminates.

(P3) Since $S'_{k_{r+i-1}}$ has $|T_i|$ pages, therefore $S'_{k_{r+i-1}} = < p'_{k_{r+i-2}+1}, p'_{k_{r+i-2}+1}, \ldots, p'_{k_{r+i-2}+|T_i|} >$. Since $T_i = F_i \cap J(d_i)$ (line 3), $D'_{k_{r+i-1}}$ contains $d_i$ and is therefore not empty. Suppose $D'_{k_{r+i-2+t}} \neq \emptyset$ for some $1 \leq t < |T_i|$. So $D'_{k_{r+i-2+t}}$ contains some page $d'$ which implies that $J(d') \cap F_i \subseteq T_i$ and $p'_{k_{r+i-2+|T_i|}} \notin J(d') \cap F_i$. Since $p'_{k_{r+i-2+|T_i|}} \in T_i$, therefore $F_i \cap J(d') \subset T_i$ which contradicts the construction of $T_i$ (line 3). Therefore, we must have $D'_{k_{r+i-2+t}} = \emptyset$ for $1 \leq t < |T_i|$. Hence $S'_{k_{r+i-1}}$ is a segment of $S$.

Suppose $S'_{k_{r+i-1}}$ is not a minimal segment of $S$. By Definition 6, $\exists d' \in D'_{k_{r+i-1}}$, such that $T_i \not\subseteq J(d')$; i.e., $\exists p' \in T_i$, such that $p' \notin J(d')$. But $d' \in D'_{k_{r+i-1}}$ implies that $F_i \cap J(d') \subseteq T_i$. Since $p' \in T_i$ and $p' \notin F_i \cap J(d')$, therefore $F_i \cap J(d') \subset T_i$ which contradicts the construction of $T_i$ (line 3). Hence $S'_{k_{r+i-1}}$ must be a minimal segment of $S'$.

(P4) By Proposition 4, $B_{j_r} = B_{j_{r-1}} - D_{j_{r-1}} \cup S_{j_r}$ and $B'_{k_r} = B'_{k_{r-1}} - D'_{k_{r-1}} \cup S'_{k_r}$. Since $S'_{k_1} \circ \ldots \circ S'_{k_{r-1}} = S_{j_1} \circ \ldots \circ S_{j_{r-1}}$, therefore

$$B'_{k_{r-1}} = B_{j_{r-1}} \text{ and } D'_{k_{r-1}} = D_{j_{r-1}}. \tag{4}$$

Consider $B'_{k_{r+i-1}}$. By Proposition 4,

$$\begin{aligned}
B'_{k_{r+i-1}} &= B'_{k_{r-1}} \cup \textstyle\bigcup_{1\leq t\leq i} S'_{k_{r+i-t}} - \bigcup_{1\leq t\leq i} D'_{k_{r+i-1-t}} \\
&= B'_{k_{r-1}} - D'_{k_{r-1}} \cup \textstyle\bigcup_{1\leq t\leq i} S'_{k_{r+i-t}} - \bigcup_{1\leq t< i} D'_{k_{r+i-1-t}} \\
&\subset B'_{k_{r-1}} - D'_{k_{r-1}} \cup \textstyle\bigcup_{1\leq t\leq i} S'_{k_{r+i-t}} \text{ (since each } D'_{k_{r+i-1-t}} \neq \emptyset) \\
&= B'_{k_{r-1}} - D'_{k_{r-1}} \cup S_{j_r} \text{ (since } S'_{k_r} \circ \ldots \circ S'_{k_{r+n-1}} \text{ is a permutation of } S_{j_r}) \\
&= B_{j_{r-1}} - D_{j_{r-1}} \cup S_{j_r} \text{ (by Equation 4)} \\
&= B_{j_r} \text{ (by Proposition 4)}
\end{aligned}$$

Hence $|B'_{k_{r+i-1}}| < |B_{j_r}|$.

Since (1) $B'_{k_i} = B_{j_i}$ for $1 \leq i < r$, (2) $|B'_{k_i}| < |B_{j_r}|$ for $r \leq i < r+n$, and (3) $B'_{k_i} = B_{j_{i-n+1}}$ for $r+n \leq i \leq m+n-1$, therefore $MAX(S') \leq MAX(S)$. In particular, if $MAX(S) = B_{j_i}$ iff $i = r$, then $MAX(S') < MAX(S)$. □

Algorithm **MINIMIZE** ($S_{j_r}$)

Input: $S_{j_r}$ is a non-minimal segment with no premature page accesses.
Output: $S'_{k_{r+i-1}}$ is a minimal segment, $1 \leq i \leq n$ for some $n \geq 1$.

$i := 1$; $F_i := \{p | p \text{ is in } S_{j_r}\}$; $A_i := D_{j_r}$;
**repeat**
    Let $T_i := F_i \cap J(d_i)$ for some $d_i \in A_i$ such that $\nexists d' \in A_i, \emptyset \subset F_i \cap J(d') \subset T_i$;
    Let $S'_{k_{r+i-1}}$ be any permutation of the pages in $T_i$;
    $F_{i+1} := F_i - T_i$;
    $A_{i+1} := \{d \in A_i | F_{i+1} \cap J(d) \neq \emptyset\}$;
    $i := i + 1$;
**until** ($F_i = \emptyset$);

Figure 2: Minimize Algorithm

**Theorem 1** *Let $S$ be a page access sequence. Then $S$ can be refined into another page access sequence $S'$ such that $S'$ is a minimal page access sequence and $MAX(S') \leq MAX(S)$.*

**Proof**: By Proposition 2, $S$ can be refined into another page access sequence $S'$ such that $S'$ is non-premature and $MAX(S') \leq MAX(S)$. By Proposition 5, $S'$ can be refined into another page access sequence $S''$ such that $S''$ is minimal and $MAX(S'') \leq MAX(S') \leq MAX(S)$. □

By Theorem 1, an arbitrary page access sequence can be refined into a minimal page access sequence with no worse upper bound. This result can be used to augment existing OPAS1 heuristics to generate minimal page access sequences with possibly lower upper bounds.

## 3 A New Heuristic for the OPAS1 Problem

We shall now present a new greedy heuristic for the OPAS1 problem, referred to as the COH heuristic, which is based on the notion of minimal segments. Since minimal page access sequences are desirable, the main idea of our new heuristic is to construct a minimal page access sequence using minimal segments. At each iteration, the heuristic fetches the next "best" minimal segment until all the pages have been fetched. The greedy algorithm is illustrated in Figure 3. The buffer is first initialized to be empty (line 1); i.e., all the join pages are initially disk resident. A page $p$ is buffer resident iff $mark(p) = true$. The algorithm iteratively selects the next "best" minimal segment (i.e., a set of pages) using a greedy heuristic **SelectSegment** (line 5) and fetches it into the buffer (line 6). The pages fetched are then joined with the other resident buffer pages (line 7). Buffer pages that are made available after the join are released (line 8). The algorithm terminates when

Algorithm **OPAS1** ($G$)

Input: $G = (V, E)$ is a connected join graph.
Output: $S = S_{j_1} \circ S_{j_2} \circ \ldots \circ S_{j_i}$ is a minimal page access sequence.

$mark(v)$ := *false* for all $v \in V$; /* Initialize buffer to be empty */
$i := 0$;
**repeat**
    $i := i + 1$;
    $S_{j_i}$ := **SelectSegment**($G$); /* Select the $i^{th}$ segment to fetch */
    $mark(p)$ := *true* $\forall p \in S_{j_i}$; /* Fetch $S_{j_i}$ into the buffer */
    Delete all the edges $(u, v) \in E$ where $mark(u) = mark(v) = true$; /* Perform join */
    Delete all the vertices in $V$ with zero degrees; /* Remove available pages */
**until** ($V = \emptyset$);

Figure 3: Algorithm of New OPAS1 Heuristic (COH)

all the join pages have been fetched. Note that each page in the join graph is fetched exactly once into the buffer since a page once brought into the buffer will remain buffer resident until it has joined with all its adjacent pages in the join graph. The page access sequence obtained is $S_{j_1} \circ S_{j_2} \circ \ldots \circ S_{j_i}$.

The **SelectSegment** heuristic selects the minimal segment that has the shortest length where ties are resolved arbitrarily. Let $J'(p)$ denote the subset of pages in the join set of a page $p$ that have not been fetched into the buffer; i.e.,

$$J'(p) = \{v \in V | v \in J(p), mark(v) = false\}.$$

The **SelectSegment** heuristic returns $J'(q)$ where

$$|J'(q)| \leq |J'(p)| \forall p \in V.$$

We claim that $J'(q)$ is the shortest minimal segment that can be selected from $V$. Consider the selection of the $i^{th}$ segment, $S_{j_i}$. For $D_{j_i} \neq \emptyset$, $S_{j_i} \supseteq J'(p)$ for some $p \in V$ so that $p \in D_{j_i}$. Suppose $S_{j_i} = J'(q)$ where $|J'(q)| \leq |J'(p)| \forall p \in V$. So $q \in D_{j_i}$. Clearly, if $S_{j_i} \subset J'(q)$, then $D_{j_i} = \emptyset$. Thus $S_{j_i}$ is a segment. Suppose $q' \in D_{j_i}$ and $q' \neq q$. This implies that $J'(q) = J'(q')$; i.e., $S_{j_i} \subseteq J(q')$. So $S_{j_i}$ is also a minimal segment. Since $|J'(q)| \leq |J'(p)| \forall p \in V$, $S_{j_i}$ is the shortest minimal segment. The time complexity of the COH heuristic is $O(|V|^2)$.

# 4 Comparison with Existing OPAS1 Heuristics

The existing OPAS1 heuristics [3, 10] [1] are based on the greedy paradigm shown in Figure 4. In contrast to our new heuristic (Figure 3) which selects a set of pages at each iteration, existing heuristics select one page at a time.

[1] We have excluded in this study an earlier heuristic by Pramanik and Ittner [11] as its performance has been compared with Omiecinski's work [10].

Algorithm **OPAS1** ($G$)

Input: $G = (V, E)$ is a connected join graph;
Output: $S = < p_1, p_2, \ldots, p_{|V|} >$ is a page access sequence.

$mark(v) := \mathit{false}$ for all $v \in V$; /* Initialize buffer to be empty */
$i := 0$;
**repeat**
    $i := i + 1$;
    $p_i :=$ **SelectPage**($G$); /* Select the next page to fetch */
    $mark(p_i) := true$; /* Fetch $p_i$ into the buffer */
    Delete all the edges $(u, v) \in E$ where $mark(u) = mark(v) = true$; /* Perform join */
    Delete all the vertices in $V$ with zero degrees; /* Remove available pages */
**until** $(V = \emptyset)$;

Figure 4: Algorithm of Existing OPAS1 Heuristics (FPH & OH)

We shall refer to the heuristic proposed by Fotouhi and Pramanik [3] as FPH and the heuristic proposed by Omiecinski [10] as OH. The FPH is designed for general join graphs while the OH is designed specially for bipartite join graphs. Both heuristics select the next "best" page based on the characteristics of the join graph. The **resident degree** of a vertex $v \in V$, denoted by $rDeg(v)$, is defined as the number of vertices adjacent to $v$ in the join graph that are resident in the buffer. The **non-resident degree** of a vertex $v \in V$, denoted by $nrDeg(v)$, is defined as the number of vertices adjacent to $v$ in the join graph that have not been fetched into the buffer. In other words, $rDeg(v) = |\{w \in V | (v, w) \in E, mark(w) = true\}|$ and $nrDeg(v) = |\{w \in V | (v, w) \in E, mark(w) = \mathit{false}\}|$. Thus, the degree of a vertex $v \in V$, denoted by $Deg(v)$, is the sum of its resident and non-resident degrees. The **SelectPage** function for both FPH and OH heuristics are shown in Figure 5.

Table 1 summarizes the comparison of our OPAS1 heuristic with existing heuristics. The main distinction between the new heuristic and existing heuristics is that

Table 1: Comparison of OPAS1 Heuristics

| | | Page Selection Heuristic | |
|---|---|---|---|
| | | One-Page-At-A-Time | Set-of-Pages-At-A-Time |
| Type of Join Graph | General Graphs | FPH [3]<br>$O(\lvert V\rvert^2)$ | COH<br>$O(\lvert V\rvert^2)$ |
| | Bipartite Graphs | OH [10]<br>$O(\lvert V_1\rvert^2 \times \lvert E\rvert^2)$, $\lvert V_1\rvert \leq \lvert V_2\rvert$ | |

the new heuristic constructs the page access sequence in terms of minimal segments

**SelectPage**$(G)$ /* FPH Heuristic */
Select $p$ to be the non-resident vertex in $V$ with the largest resident degree;
If there is more than one such vertex, then choose the one with
the smallest non-resident degree;
return $p$;

**SelectPage**$(G)$ /* OH Heuristic */
**if** the buffer is empty **then**
**begin**
Select $(p,q) \in E$ such that $Deg(p) + Deg(q)$ is the smallest;
$mark(q)$ := $true$; /* Fetch $q$ into the buffer */
**end else begin**
Let $X$ be the subset of resident vertices in $V$ such that each vertex
has the smallest non-resident degree;
Select $p$ to be the non-resident vertex in $V$ that has the smallest
non-resident degree for all $(p,x) \in E, x \in X$;
**end**
return $p$;

Figure 5: Existing OPAS1 Heuristics

instead of single pages. Thus the page access sequence produced by the new heuristic is guaranteed to be minimal, which as we have shown is desirable. It is not clear whether the page access sequences generated by existing heuristics have this property.

# 5 Performance Comparison of OPAS1 Heuristics

This section compares the performance of the new heuristic COH with existing OPAS1 heuristics. The performance experiments are characterized by the following parameters: the type of join graphs used (general or bipartite), size of the join relations in terms of number of pages, the type of heuristics applied, and the edge ratio of the join graphs. Given a join graph $G = (V, E)$, the **edge ratio of** $G$ is defined to be the ratio of the total number of edges in $G$ to the maximum possible number of edges in $G$; i.e., in a fully connected graph.

$$\text{Edge Ratio of } G = \begin{cases} \frac{2|E|}{|V|(|V|-1)} & \text{if } G \text{ is a general join graph} \\ \frac{|E|}{|V_1||V_2|} & \text{if } G \text{ is a bipartite join graph} \end{cases}$$

The edge ratio provides an approximate measure of the join selectivity.

By varying the join graph type, size of join relations, and edge ratio, different classes of join graphs are obtained. For each class of join graphs, 20 random instances are created, and the optimal buffer size given by the different OPAS1 heuristics measured. Let $\overline{B}(h)$ denote the mean optimal buffer size of the 20 random join graphs obtained using heuristic $h$.

The results for general join graphs are shown in Table 2. The COH heuristic clearly outperforms the FPH heuristic for all the classes of join graphs shown. For a given join graph size (i.e. $|V|$), the performance gain of the COH heuristic over the FPH heuristic decreases as the edge ratio increases.

Table 2: Comparison of OPAS1 Heuristics for General Join Graphs

| $\|V\|$ (pages) | Edge Ratio (%) | $\overline{B}(FPH)$ (pages) | $\overline{B}(COH)$ (pages) | $\overline{B}(FPH)$ - $\overline{B}(COH)$ (pages) |
|---|---|---|---|---|
| 500 | 5 | 396.7 | 341.6 | 55.1 |
| | 10 | 443.8 | 403.5 | 40.3 |
| | 15 | 461.1 | 430.2 | 30.9 |
| | 20 | 471.5 | 446.2 | 25.2 |
| | 25 | 477.6 | 455.9 | 21.6 |
| | 30 | 481.9 | 463.5 | 18.4 |
| 1000 | 5 | 883.0 | 801.2 | 81.8 |
| | 10 | 938.9 | 883.8 | 55.1 |
| | 15 | 958.4 | 918.8 | 39.6 |
| | 20 | 968.4 | 937.0 | 31.4 |
| | 25 | 975.4 | 948.8 | 26.6 |
| | 30 | 980.2 | 958.2 | 22.0 |
| 2000 | 5 | 1872.8 | 1760.7 | 112.2 |
| | 10 | 1933.2 | 1866.6 | 66.7 |
| | 15 | 1955.7 | 1906.9 | 48.8 |
| | 20 | 1966.4 | 1929.2 | 37.2 |
| | 25 | 1974.0 | 1943.2 | 30.8 |
| | 30 | 1978.9 | 1953.1 | 25.8 |

The results for bipartite join graphs are shown in Table 3 for cases when (a) $|V_1| = |V_2|$ and (b) $|V_1| < |V_2|$. For both cases, the COH heuristic outperforms the FPH heuristic with the performance margin increasing with the edge ratio. Comparing the COH and OH heuristics, the COH heuristic outpeforms the OH heuristic when $|V_1| = |V_2|$, with the performance margin narrowing as the edge ratio increases. A possible explanation for the decreasing performance margin as the edge ratio increases is that in the OH heuristic, the **SelectPage** function (Figure 5) always selects a page that is adjacent to some resident buffer page (unless the buffer is empty); on the other hand, in the COH heuristic, the set of pages selected by the **SelectSegment** function (section 3) need not be adjacent to any resident buffer pages. In other words, the set of candidate pages available for selection for the OH heuristic is generally more restrictive than for the COH heuristic. However, as the edge ratio increases, this difference becomes smaller resulting in a smaller performance gain. When $|V_1| < |V_2|$, the optimal buffer sizes for both the heuristics are comparable, with a difference of at most one buffer page.

As noted in Section 2, if $S$ is an optimal page access sequence, $MAX(S)$ is bounded by $\min\{|V_1|, |V_2|\} + 1$. Among the three OPAS1 heuristics, the optimal buffer sizes obtained using the FPH heuristic exceed the bound for all cases. The OH heuristic exceeds the bound for some cases when $|V_1| = |V_2|$. Only the COH heuristic stays within the bound for all cases.

# 6 Heuristics for the OPAS2 Problem

In this section, we examine heuristics for the OPAS2 problem which is to determine for a given join graph and a fixed buffer size, the least upper bound on the number of page reaccesses.

The existing heuristics for this problem are simply extensions of the OPAS1 heuristics to handle page fetches when the buffer becomes full. That is, the algorithm in Figure 4 is extended with another heuristic **selectVictim** to select a victim page for replacement when an incoming page cannot be accomodated in the buffer. The extension of the algorithm in Figure 4 is shown in Figure 6. The buffer is first

---

Algorithm **OPAS2** $(G, M)$

Input: $G = (V, E)$ is a connected join graph.
$M$ is the available buffer size.
Output: $S = < p_1, p_2, \ldots, p_r >$ is a page access sequence ($p_i$'s need not be distinct) and $r \geq |V|$.
$R$ is a sequence of victim pages selected for replacement.

```
/* Steps 1 to 9 are identical to the OPAS1 algorithm (Figure 4) */
5.1)    if (|{v ∈ V|mark(v) = true}| = M) then /* buffer is full */
5.2)    begin
5.3)        q := SelectVictim(G,p_i); /* Select victim page */
5.4)        mark(q) := false; /* q is replaced; need to be fetched again */
5.5)    end
```

Figure 6: Algorithm of Existing OPAS2 Heuristics (FPH & OH)

---

checked to see if it is already full (line 5.1) before fetching the next selected page $p$. If so, another heuristic **SelectVictim** (line 5.3) is used to select the "best" resident buffer page (known as the victim page) to be replaced by $p$. The victim page needs to be fetched again subsequently (line 5.4) since it still has not joined with all its adjacency pages in the join graph (as implied by its residency in the buffer).

For the FPH heuristic [3], the victim page is the resident page with the smallest non-resident degree. For the OH heuristic [10], the victim page is the resident page that (a) has the smallest non-resident degree, and (b) is not connected to the new page to be fetched. However, if all the pages in the buffer are connected to the new page, then only criterion (a) determines the victim page. For both heuristics, the rationale for selecting the resident page with the smallest non-resident degree as the victim page is based on a pessimistic view of the buffer situation: in the worst case, the number of times that a replaced page needs to fetched again into the buffer is equal to the number of non-resident pages that it is connected to.

We have also extended the COH heuristic to handle page replacements for the OPAS2 problem as shown in Figure 7. The algorithm for the OPAS1 problem

---

Algorithm **OPAS2** $(G,M)$

Input: $G = (V, E)$ is a connected join graph.
$M$ is the available buffer size.

Output: $S = < p_1, p_2, \ldots, p_r >$ is a page access sequence ($p_i$'s need not be distinct) and $r \geq |V|$.
$R$ is a sequence of victim pages selected for replacement.

```
mark(v) := false for all v ∈ V; /* Initialize buffer to be empty */
i := 0; k := 0;
repeat
    i := i + 1;
    S_{j_i} := SelectSegment(G); /* Select the i^{th} segment to fetch */
    repeat
        k := k + 1;
        p_k := SelectPage(G,S_{j_i});
        if (|{v ∈ V|mark(v) = true}| = M) then /* buffer is full */
        begin
            q := SelectVictim(G,p_k); /* Select victim page */
            mark(q) := false; /* q is replaced; need to be fetched again */
        end
        mark(p_k) := true; /* Fetch p_k into the buffer */
        Delete all the edges (u, v) ∈ E where mark(u) = mark(v) = true; /* Perform join */
        Delete all the vertices in V with zero degrees; /* Remove available pages */
        Delete p_k from S_{j_i};
    until (S_{j_i} = ∅);
until (V = ∅);
```

Figure 7: Algorithm of New OPAS2 Heuristic (COH)

---

(Figure 3) has been extended with two additional heuristics, **SelectPage** and **SelectVictim**. Unlike in the OPAS1 problem, where only the sequence of fetching segments matters, the sequence of page access within each selected segment is also important in the OPAS2 problem. This is because the buffer size is now bounded which means that when the available buffer pages is less than the number of pages in the selected segment, some buffer pages will have to be replaced (and later fetched again) as pages from the segment are brought into the buffer. Thus the sequence of page access from each segment makes a difference to the total number of page reaccesses. Consequently, a heuristic is also needed to select the victim page to be replaced.

To select the next page to fetch from a segment, the **SelectPage** heuristic chooses the page with the largest resident degree. The intuition for this is to perform as many joins as possible for each page fetched into the buffer so as to minimize the number of subsequent page reaccesses. To resolve ties, the page with the smallest non-resident degree is selected. For the selection of victim pages for replacement, the **SelectVictim** heuristic selects the page with the smallest non-resident degree, which is similar to existing heuristics.

# 7 Performance Comparison of OPAS2 Heuristics

In this section, we compare the performance of the extended heuristics for the OPAS2 problem. The experiment parameters includes all the parameters in the previous comparison of the OPAS1 heuristics: type of join graphs, size of join relations, type of heuristics, and edge ratio of join graphs. An additional parameter is the buffer size (in terms of number of pages).

To limit the number of experiments, the edge ratio parameter is fixed to be 10%. For each class of join graphs (characterized by the join graph type and size of join relations), 20 random join graphs are created and the number of page reaccesses measured for the different OPAS2 heuristics and buffer sizes. We compare the performance of two different OPAS2 heuristics based on the the difference of their mean number of page reaccesses (for 20 join graphs). Let $\overline{R}(h)$ denote the mean number of buffer page reaccesses using heuristic $h$.

The comparison of the COH heuristic against the FPH heuristic for general join graphs is shown in Figure 8. For each graph, the *Bopt* value denotes the optimal buffer size using the COH heuristic; i.e., the buffer size when $\overline{R}(COH) = 0$. As expected, as the buffer size increases from *Bopt*, the difference in the number of page reaccesses decreases and will eventually reduce to zero. As the buffer size decreases from *Bopt*, the performance gain gradually increases to a peak and then decreases.

Figures 9 and 10 show the results for bipartite join graphs with $|V_1| = |V_2|$, comparing heuristic COH against heuristics FPH and OH, respectively. These graphs exhibit the same trend as those comparing the general join graphs in Figure 8. When the buffer size is reduced below a certain threshold, the other OPAS2 heuristics overtake the COH heuristic.

# 8 Conclusions

In this paper, we examined the issue of scheduling page access in join processing. In particular, two related problems were addressed: (1) the determination of an optimal page access sequence such that the join can be computed without any page reaccesses using the minimum number of buffer pages, and (2) the determination of an optimal page access sequence such that the join can be computed with the minimum number of page reaccesses given a limited number of buffer pages. These problems (OPAS1 and OPAS2, respectively) arise in many join evaluation strategies based on the scan-join algorithm, and are critical to the performance of the database system as they affect both buffer utilization as well as disk I/O cost. Unfortunately, both problems are NP-hard.

By modeling a page access sequence as a concatenation of segments, we derived two desirable properties of a page access sequence namely, that it should not contain premature page accesses and that it should be a minimal page access sequence. Based on these properties, we proposed a new heuristic (COH heuristic) for the OPAS1 problem that produces page access sequences that satisfy both properties; we also extended the new heuristic for the OPAS2 problem. For the OPAS1 problem, the new heuristic performs better than existing heuristics; while for the OPAS2 problem, the new heuristic's performance is better than existing heuristics provided that the number of available buffer pages is not too much less than the optimal buffer size. Furthermore, compared to the OH heuristic, the COH heuristic has a lower time complexity bound.

# Acknowledgement

We would like to thank Wong Lim Soon for reading an earlier draft of this paper.

# References

[1] E. Bertino and W. Kim. Indexing techniques for queries on nested objects. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):196–214, June 1989.

[2] B.C. Desai. Performance of a composite attribute and join index. *IEEE Transactions on Software Engineering*, 15(2):142–152, February 1989.

[3] F. Fotouhi and S. Pramanik. Optimal secondary storage access sequence for performing relational join. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):318–328, September 1989.

[4] G. Graefe. Query evaluation techniques for large databases. *Computing Surveys*, 25(2):73–170, 1993.

[5] A. Kemper and G. Moerkotte. Access support in object bases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 364–374, Atlantic City, N.J., 1990.

[6] D. Maier and J. Stein. Indexing in an object-oriented dbms. In *Proceedings of International Workshop on Object-Oriented Database Systems*, pages 171–182, Asilomar, California, September 1986. Also in *On Object-Oriented Database Systems*, edited by Dittrich, K.R., Dayal, U. and Buchmann, A.P., Chapter 20, Springer-Verlag, 1991.

[7] T. Merrett, Y. Kambayashi, and H. Yasuura. Scheduling of page fetches in join operations. In *Proceedings of 7th International Conference on Very Large Data Bases*, pages 488–498, Cannes, France, September 1981.

[8] P. Mishra and M.H. Eich. Join processing in relational databases. *Computing Surveys*, 24(1):63–113, 1992.

[9] R. Ng, C. Faloutsos, and T. Sellis. Flexible buffer allocation based on marginal gains. In *Proceedings of ACM SIGMOD Conference*, pages 387–396, Denver, Colorado, May 1991.

[10] E.R. Omiecinski. Heuristics for join processing using nonclustered indexes. *IEEE Transactions on Software Engineering*, 15(1):18–25, January 1989.

[11] S. Pramanik and D. Ittner. Use of graph-theoretic models for optimal relational database accesses to perform join. *ACM Transactions on Database Systems*, 10(1):57–74, March 1985.

[12] P. Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2):218–246, June 1987.

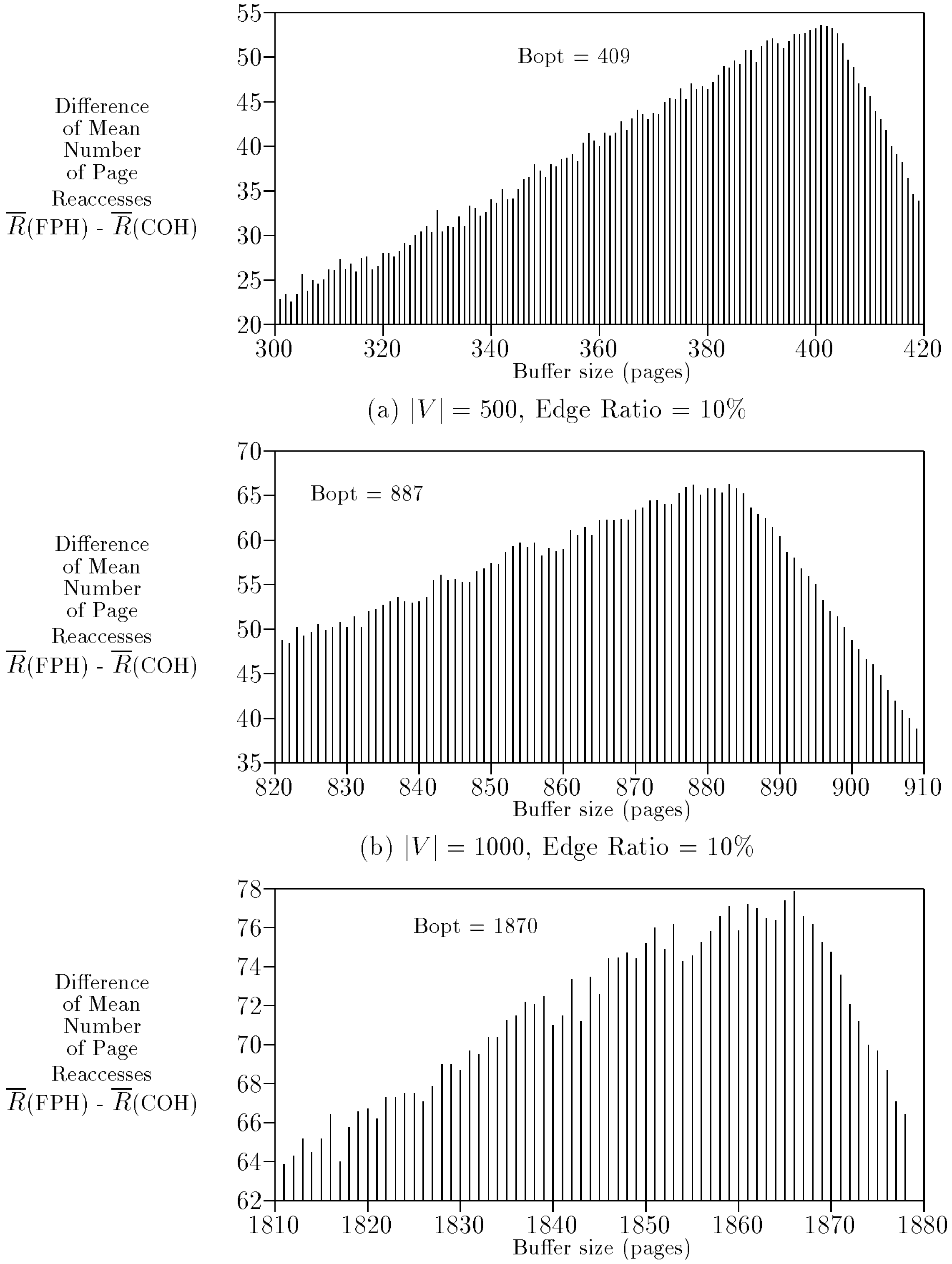
Table 3: Comparison of OPAS1 Heuristics for Bipartite Join Graphs

(a) $|V_1| = |V_2|$

| $\|V_1\|$ (pages) | $\|V_2\|$ (pages) | Edge Ratio(%) | $\overline{B}(FPH)$ (pages) | $\overline{B}(OH)$ (pages) | $\overline{B}(COH)$ (pages) | $\overline{B}(FPH)$ - $\overline{B}(COH)$ | $\overline{B}(OH)$ - $\overline{B}(COH)$ |
|---|---|---|---|---|---|---|---|
| 250 | 250 | 5 | 323.2 | 243.6 | 232.2 | 91.1 | 11.4 |
| | | 10 | 398.8 | 250.4 | 247.2 | 151.6 | 3.2 |
| | | 15 | 429.1 | 251.5 | 249.9 | 179.2 | 1.6 |
| | | 20 | 445.7 | 252.1 | 250.8 | 194.9 | 1.2 |
| | | 25 | 456.8 | 251.6 | 251.0 | 205.8 | 0.6 |
| | | 30 | 464.1 | 251.0 | 251.0 | 213.1 | 0.0 |
| 500 | 500 | 5 | 792.9 | 500.4 | 491.6 | 301.2 | 8.8 |
| | | 10 | 885.5 | 501.8 | 499.6 | 385.9 | 2.2 |
| | | 15 | 921.8 | 503.0 | 500.9 | 420.9 | 2.1 |
| | | 20 | 941.6 | 501.0 | 501.0 | 440.6 | 0.0 |
| | | 25 | 952.8 | 501.0 | 501.0 | 451.8 | 0.0 |
| | | 30 | 961.8 | 501.0 | 501.0 | 460.8 | 0.0 |
| 1000 | 1000 | 5 | 1769.0 | 1014.3 | 996.9 | 772.1 | 17.4 |
| | | 10 | 1874.3 | 1008.9 | 1000.6 | 873.7 | 8.3 |
| | | 15 | 1914.8 | 1003.9 | 1001.0 | 913.8 | 2.9 |
| | | 20 | 1935.8 | 1001.0 | 1001.0 | 934.8 | 0.0 |
| | | 25 | 1949.0 | 1001.0 | 1001.0 | 948.0 | 0.0 |
| | | 30 | 1959.0 | 1001.0 | 1001.0 | 958.0 | 0.0 |

(b) $|V_1| < |V_2|$

| $\|V_1\|$ (pages) | $\|V_2\|$ (pages) | Edge Ratio(%) | $\overline{B}(FPH)$ (pages) | $\overline{B}(OH)$ (pages) | $\overline{B}(COH)$ (pages) | $\overline{B}(FPH)$ - $\overline{B}(COH)$ | $\overline{B}(OH)$ - $\overline{B}(COH)$ |
|---|---|---|---|---|---|---|---|
| 200 | 400 | 5 | 323.6 | 190.7 | 190.4 | 133.2 | 0.2 |
| | | 10 | 373.1 | 199.2 | 199.1 | 174.1 | 0.1 |
| | | 15 | 392.5 | 200.7 | 200.8 | 191.7 | $-0.2$ |
| | | 20 | 402.5 | 201.0 | 201.0 | 201.5 | 0.0 |
| | | 25 | 408.1 | 201.0 | 201.0 | 207.1 | 0.0 |
| | | 30 | 410.8 | 201.0 | 201.0 | 209.8 | 0.0 |
| 200 | 600 | 5 | 348.8 | 193.9 | 194.8 | 153.9 | $-0.9$ |
| | | 10 | 391.9 | 199.8 | 200.0 | 191.9 | $-0.2$ |
| | | 15 | 409.9 | 201.0 | 201.0 | 208.9 | 0.0 |
| | | 20 | 416.6 | 201.0 | 201.0 | 215.6 | 0.0 |
| | | 25 | 420.7 | 201.0 | 201.0 | 219.7 | 0.0 |
| | | 30 | 421.9 | 201.0 | 201.0 | 220.9 | 0.0 |
| 200 | 800 | 5 | 363.6 | 196.2 | 196.7 | 167.0 | $-0.4$ |
| | | 10 | 405.4 | 200.2 | 200.3 | 205.0 | $-0.1$ |
| | | 15 | 420.6 | 201.0 | 201.0 | 219.6 | 0.0 |
| | | 20 | 426.6 | 201.0 | 201.0 | 225.6 | 0.0 |
| | | 25 | 427.1 | 201.0 | 201.0 | 226.1 | 0.0 |
| | | 30 | 430.3 | 201.0 | 201.0 | 229.3 | 0.0 |

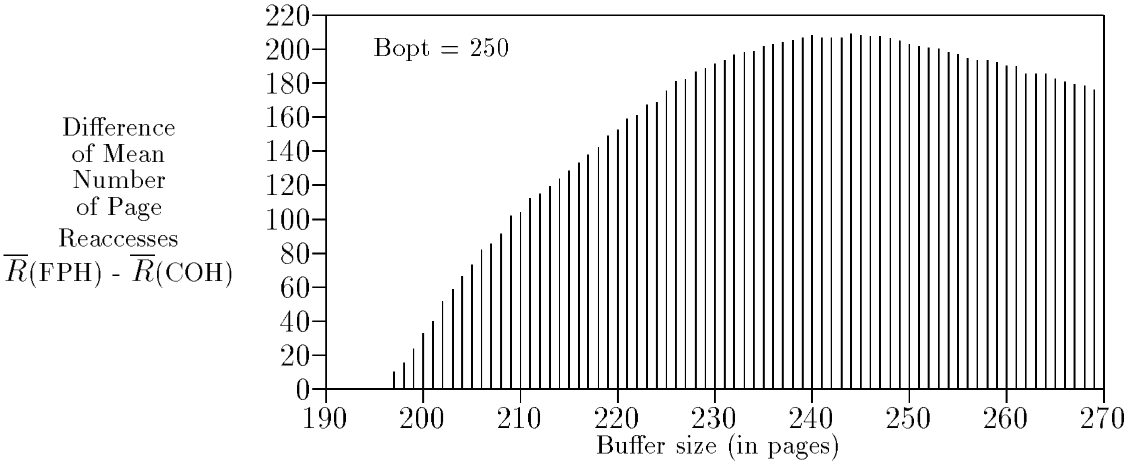


(a) $|V| = 500$, Edge Ratio = 10%

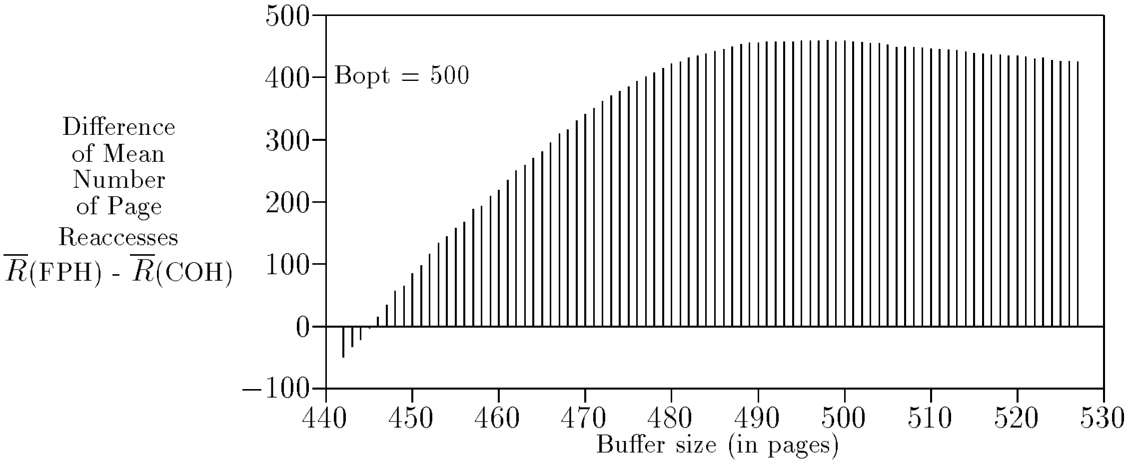

(b) $|V| = 1000$, Edge Ratio = 10%


(c) $|V| = 2000$, Edge Ratio = 10%

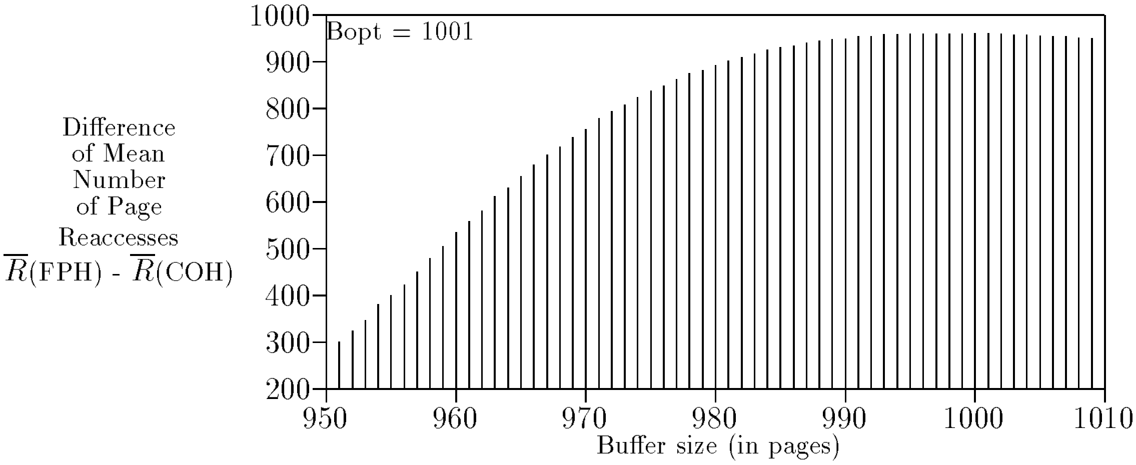
Figure 8: Comparison of OPAS2 Heuristics (FPH vs COH) for General Join Graphs




(a) $|V_1| = 250, |V_2| = 250$, Edge Ratio = 10%

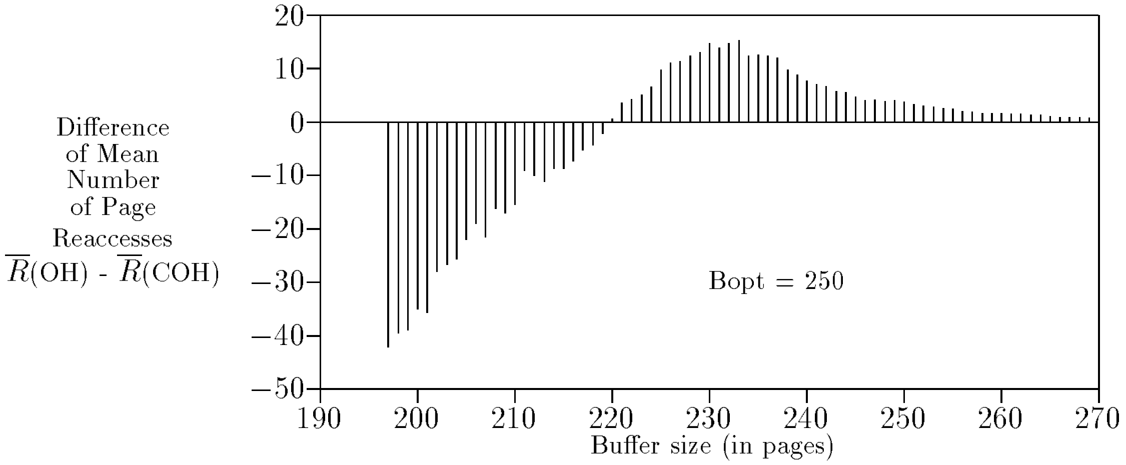



(b) $|V_1| = 500, |V_2| = 500$, Edge Ratio = 10%

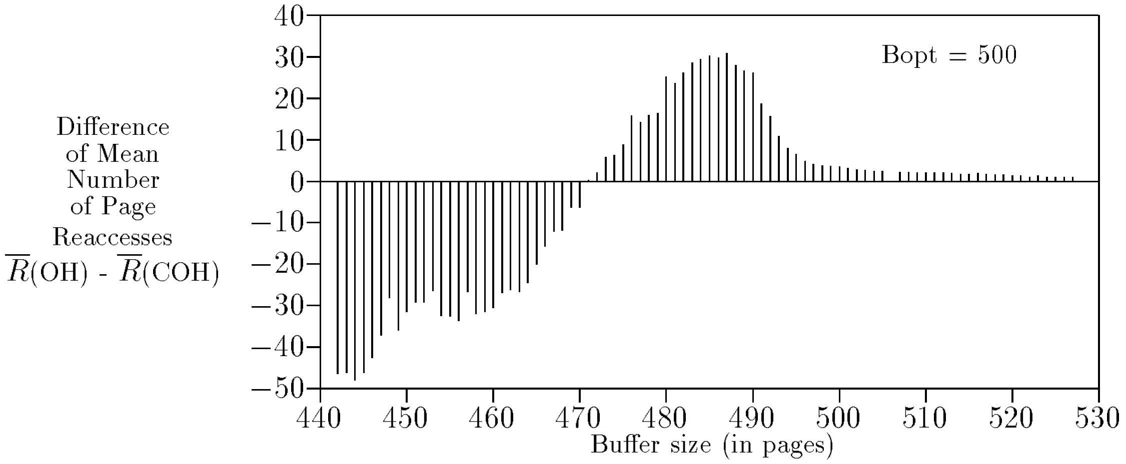



(c) $|V_1| = 1000, |V_2| = 1000$, Edge Ratio = 10%

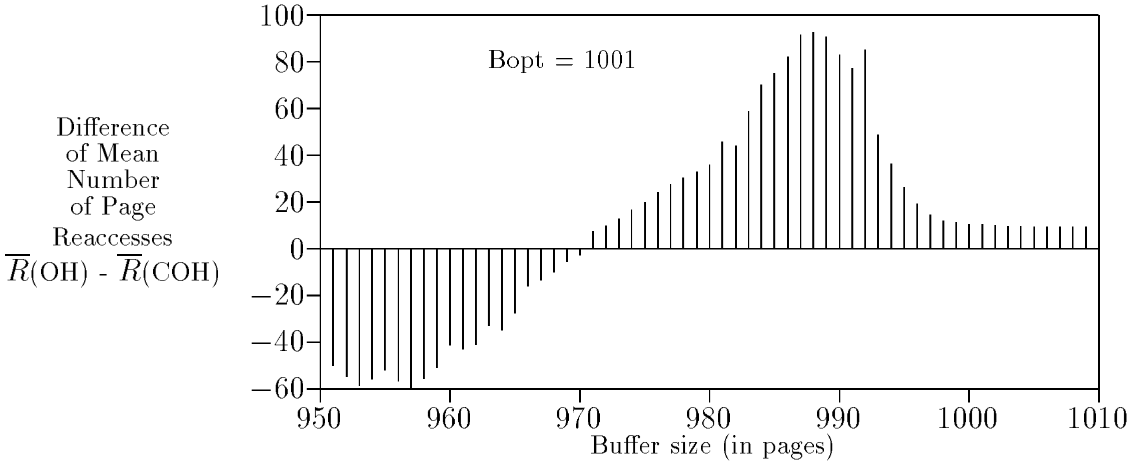
Figure 9: Comparison of OPAS2 Heuristics (FPH vs COH) for Bipartite Join Graphs




(a) $|V_1| = 250, |V_2| = 250$, Edge Ratio = 10%




(b) $|V_1| = 500, |V_2| = 500$, Edge Ratio = 10%




(c) $|V_1| = 1000, |V_2| = 1000$, Edge Ratio = 10%

Figure 10: Comparison of OPAS2 Heuristics (OH vs COH) for Bipartite Join Graphs