Jagged Bite Problem NP-Complete Construction

Kris Hildrum and Megan Thomas
Computer Science Division
University of California, Berkeley
{hildrum,mct}@cs.berkeley.edu

ABSTRACT

A hyper-rectangle is commonly used as a bounding predicate in tree-based access methods in databases. To reduce the number of I/O's per query, we would like to reduce the volume of this bounding predicate by cutting chunks out of the corners of the bounding hyper-rectangle. Ideally, one would like to remove the largest hyper-rectangular chunk that does not contain any points. In this paper, we formalize the problem and then show that the problem of finding the largest possible chunk is NP-complete. We accomplish this through a reduction from 3-SAT to our problem.

The Jagged-Bites Problem

The Jagged-Bites problem, defined after we provide some background, is significant to a database access method designer working with nearest neighbor queries because of the way that such queries interact with rectangular bounding predicates (BPs). It was first suggested in [1].

Nearest neighbor queries work by finding points within a given distance of the query point, in essence asking expanding hyper-sphere queries. Figure 1 depicts a rectangular, 2-D BP and the circles of nearest neighbor queries. If the query point is in the middle of a BP, it does not matter how small the BP volume is; this node will be accessed. However, nearest neighbor queries starting from points just outside a particular BP may or may not intersect that BP. For good performance on nearest neighbor query workloads, the intersection of BPs with non-matching query spheres, like the two topmost queries in Figure 1, must be minimized.

The data points of R-tree[1] leaf nodes do not always fill their minimum bounding rectangles (MBRs), instead leaving noticeable gaps at corners of the MBRs. This suggests that query execution costs may rise while checking the contents of the empty corners of the MBR, as the two top queries in Figure 1 do. Combined with our intuition about the importance of reducing spherical intersections with BP regions near the BP edges, this leads us to focus on attempting to remove empty areas from the corners of an MBR BP, by "biting" into the volume of the BP from the corners.

Consider the points in the rectangle in Figure 1. A two-rectangle BP, pictured in Figure 2, can bind those same data points more tightly than the MBR BP. However, the two-rectangle BP still leaves empty areas at some corners of each rectangle of the BP in this figure. This would be less problematic in a BP that stored the MBR and the largest possible rectangular "bite" taken out of each corner. This type of BP, which we call a "Jagged Bites" BP (JB), would describe the data in a node more precisely than a two rectangle BP. The JB BP, pictured in Figure 3 for the same set of data points, stores the MBR of a set of data, as well as a set of points identifying the bites. Just as the MBR of a data set can be represented by storing two points, one for the highest values in each dimension, and one for the lowest, a corner bite

[1] An R-tree is a balanced tree-structured database access method for multi-dimensional data. Data is stored at the leaf level of the tree, while upper levels contain minimum bounding rectangles and pointers to the corresponding sets of data points (or MBRs, in the case of multilevel trees) that the MBRs bound. Query execution cost, measured in I/O's, increases as the number of leaf level pages read into memory in order to check for data that actually satisfies the query increases.

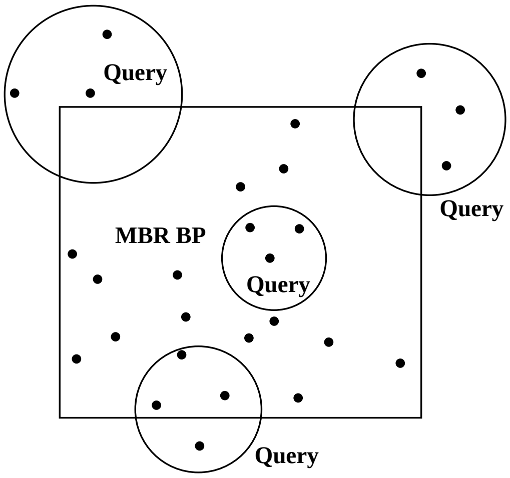

Figure 1: Nearest Neighbor Queries And Rectangular BPs

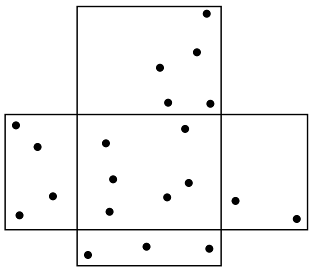

Figure 2: A Two-Rectangle BP

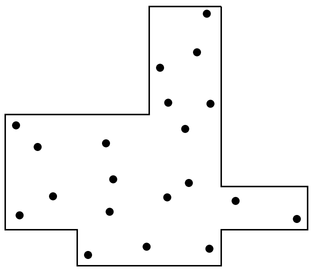

Figure 3: A Jagged Bites BP

can be represented by the associated MBR corner point and another point at the one "internal" corner of the bite rectangle, which might not intersect any MBR hyper edge.

Here we prove that a polynomial-time algorithm for constructing an optimal JB bounding predicate is not possible unless $P = NP$, so a heuristic JB BP construction algorithm is necessary.

Definitions

Without loss of generality, we assume that all points are positive, and that the MBR corner we bite into will be located at $\vec{0}$. If this is not the case, the problem can be adjusted by shifting and reflection so that it holds.

We also assume that all the p_i in all the points $\vec{p}$ are finite, i.e., $\forall p_i$ in all $\vec{p}, p_i < C$ for some finite number C.

JB problem

JB (jagged-bite) problem:
Given a set P of n-dimensional points, maximize

$$\prod_{i=0}^{n-1} v_i = v_0 v_1 \dots v_{n-1}$$

subject to $\forall \vec{p} \in P, \exists i$ such that $p_i \geq v_i$.

JBP problem

To argue NP-hardness, we need to turn the above problem into a problem that has a yes/no answer. Call this problem the JBP (jagged-bite-predicate).

JBP problem:
Given a number k, and a set of points P, does there exist a vector $\vec{v}$ such that:

- $\prod v_i \geq k$
- $\forall \vec{p} \in P, \exists i$ such that $p_i \geq v_i$?

Notice that these two problems are polynomial time equivalent. It is clear that solving jagged-bite problem means one can solve the predicate version. To go the other way, use JBP as an oracle in a binary search to find the maximum k, which gives an answer to the JB problem. For example, start with $k = 100$, and if the answer is yes, double k. (Otherwise, halve k.) If the answer is no, then set $k = 1/2(100 + 200)$, then if the answer is yes, set $k = 175$ and so on until k is found. This will take a logarithmic number of uses of the JBP oracle.

3-SAT

Given variables $x_0 \dots x_{n-1}$ and a formula $\phi = c_1 \wedge c_2 \wedge \dots c_m$, where c_i is the disjunction of three literals (for example $c = x_3 \vee \neg x_2 \vee x_5$), determine whether ϕ is satisfiable.

JBP is NP-hard

We show that a polynomial-time solution of the JBP problem provides a polynomial-time solution to the 3-SAT problem. Therefore, JBP is NP-hard.

The Reduction

Given the formula ϕ, we create the following JBP problem by creating points of dimension $2n$ and setting k.

Intuitively, our reduction will relate $\vec{v}$ to a satisfying assignment. The vector $\vec{v}$ will have a location in it for each literal (a literal is a variable or its negation). Loosely, $\vec{v} = (x_0, \neg x_0, x_1, \neg x_1, \ldots, \neg x_{n-1})$, where $x_i = 1$ if x_i is true, and $x_i = 2$ if x_i is false. Likewise, $\neg x_i$ is 2 if x_1 is true, and $\neg x_1$ is 1 if x_i is false. In the discussion that follows, v_{2i} corresponds to the literal x_i, and v_{2i+1} corresponds to $\neg x_i$.

In order for the above description of $\vec{v}$ to make sense, certain values of $\vec{v}$ must be prevented. For example $\vec{v} = (1, 1, 1, 2, 1, 2)$ would imply that both x_0 and $\neg x_0$ are true, which would not correctly model the satisfiability problem. Some of the points created in the reduction exist only to force $\vec{v}$ to be well-formed. Other points are created from the clauses of ϕ such that satisfying a clause is equivalent to finding an i such that $p_i \geq v_i$ for the corresponding point.

As a running example to clarify the procedure, we will use: $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

clause points Convert a clause c of ϕ to a point $\vec{p}$ of dimension $2n$ in the following way.
$p_{2i} = 1$ if x_i appears in clause c, $p_{2i+1} = 1$ if $\neg x_i$ appears in clause c, and both are 0 otherwise. Create one such point for each clause of ϕ. The purpose of the clause points is to capture the particular formula ϕ. The clauses in our running example become the points $(1, 0, 1, 0, 1, 0)$ and $(0, 1, 0, 1, 0, 1)$.

two-points For every $i \in [0, 2n-1]$ we add the point $p_j = \begin{cases} 2 & j = i \\ 0 & otherwise \end{cases}$.

The purpose of the two-points is to ensure that all the v_i will be ≤ 2. In other words, to make the JB hyper-rectangle into a hyper-square with all sides of length 2. This means, for example, that a vector $\vec{v} = (1, 1, 1, 2^4)$ will not be possible. The example formula generates $(2, 0, 0, 0, 0, 0)$, $(0, 2, 0, 0, 0, 0)$, $(0, 0, 2, 0, 0, 0)$, $(0, 0, 0, 2, 0, 0)$, $(0, 0, 0, 0, 2, 0)$, and $(0, 0, 0, 0, 0, 2)$.

one-points For every $i \in [0, n-1]$ we add the point $p_j = \begin{cases} 1 & j \in [2i, 2i+1] \\ 0 & otherwise \end{cases}$ This means the points $(1, 1, 0 \ldots 0), (0, 0, 1, 1, 0, \ldots, 0)$ and so on. These points prevent a variable and its negation from being false at the same time. Indirectly, this prevents both a variable and its negation from being true. In terms of points, one-points prevent $\vec{v}$ from being $(2, 2, 1, 2, \ldots)$

The example formula generates $(1, 1, 0, 0, 0, 0)$, $(0, 0, 1, 1, 0, 0)$, and $(0, 0, 0, 0, 1, 1)$.

Choose $k = 2^n$. Though listed last, this condition is vital. Combined with one-points and two-points, this is the condition that forces $\vec{v}$ to have the desired structure of 1's and 2's.

The example formula $k = 2^3 = 8$.

In the next two sections, we will prove that our reduction is correct. That is, we must show that ϕ is satisfiable if and only if JBP says yes.

ϕ satisfiable $\Rightarrow$ JBP says yes

We want to show that if the 3-SAT formula is satisfiable, then the JBP algorithm will say yes. To prove this this, we will find a $\vec{v}$ such that:

- $\prod v_i \geq 2^n$
- For all points $\vec{p}$, $\exists i$ s.t. $p_i \geq v_i$

For the ϕ given above, one satisfing assignment is $x_1 = true$, $x_2 = false$, and $x_3 = false$. From this assignment, the proof below would construct the $\vec{v} = (1, 2, 2, 1, 2, 1)$.

Proof

Choose $v_{2i} = 1$ if x_i is true, 2 otherwise. Similarly, $v_{2i+1} = 2$ if x_i is true, 1 otherwise. Clearly, $\prod v_i = 2^n \geq k = 2^n$, so the first condition is satisfied.

Now, we must show that for all points $\vec{p}$, $\exists j$ such that $p_j \geq v_j$. There are three cases:

- $\vec{p}$ is a clause point. We need to show that in one of the variables with a 1, $\vec{v}$ also has a one. Fortunately, we know that at least one literal in the clause corresponding to that point is true. Suppose that literal is x_i (or $\neg x_i$). Then $v_{2i} = 1$, (or $v_{2i+1} = 1$) since we constructed $\vec{v}$ to ensure that this would be true. Therefore, we have that $p_{2i} \geq v_{2i}$ (or $p_{2i+1} \geq v_{2i+1}$).

- $\vec{p}$ is of the form $(\ldots 0,\ 2,\ \ldots 0)$ We selected $\vec{v}$ such that every element of $\vec{v}$ is less than or equal to 2, so this holds.

- $\vec{p}$ is of the form $(\ldots,1,\ 1,\ 0,\ \ldots 0)$ We constructed $\vec{v}$ such that one of every pair is 1, so this is satisfied.

JBP says yes $\Rightarrow$ ϕ satisfiable

Suppose that the JBP says yes. From the vector $\vec{v}$, we will construct a satisfying assignment to ϕ.

First, recall that we know that all v_i are less than 2 (because of the two-points). Further, for every pair v_{2i}, v_{2i+1}, we know that at least one of those pairs is less than one (because of the one-points). So, we know that $v_{2i} \times v_{2i+1} \leq 2$. Finally, since we know that $\prod v_i \geq 2^n$, we can conclude that $v_{2i} \times v_{2i+1} = 2$. Since each of them has a maximum of two, we can conclude that of the pair v_{2i}, v_{2i+1}, exactly one must be 2, and the other must be 1.

So, we construct a satisfying assignment in the following way: x_i is true if $v_{2i} = 1$, and x_i is false if $v_{2i+1} = 1$.

Now we show that for this assignment, each clause has at least one true literal, and so the formula is satisfied. Fix a clause c. This clause corresponds to a point $\vec{p}$, and we know that for some i, $p_i \geq v_i$. The only way $p_i \geq v_i$ is if $p_i = v_i = 1$. (Since p_i can be 0 or 1, and v_i can only be 1 or 2.) If v_i is 1, then that means the corresponding literal is true, and if p_i is 1, then that means that the corresponding literal is in the the clause c, which means that c contains at least one true literal, so c is satisfied. Since this holds for every c, the formula as a whole is satisfied.

NP-completeness

A problem is NP-complete if it is is NP-hard and in NP. We have shown that the JBP problem is NP-hard, so to show it is NP-complete, we need only show that it is in NP.

A problem is in NP if it has a polynomial size witness that can be verified in polynomial time. In this case, $\vec{v}$ is a witness. It is easy to check, given $\vec{v}$, that $\prod v_i \geq k$ and that $\forall \vec{p}, \exists i$ such that $p_i \geq v_i$. This means that JBP is NP-complete.

References

[1] M. Thomas, C. Carson, and J. Hellerstein. Creating a Customized Access Method for Blobworld. Submitted to ICDE 2000.