The 3sum problem admits subquadratic solutions

Course in Advanced Algorithms

BORELLI ROBERTO

borelli.roberto@spes.uniud.it

University of Udine

July 25, 2023

Abstract

In this work, I consider the 3SUM problem. Recent years' studies have shown that the problem admits a subquadratic solution. The 3SUM problem has been used in the area of fine-grained complexity to establish lower bounds to a wide range of other problems (which have shown to be 3SUM-hard) for example in the computational geometry area. In this paper, I examine the Freund approach to obtain a subquadratic algorithm. To obtain a saving in the complexity several tricks have been applied and in particular it has been shown how to efficiently enumerate the so-called chunks through a correspondence with paths in a matrix and then all pairs of blocks agreeing with such derived chunks are obtained through a reduction to the DOMINANCE-MERGE problem.

Contents

Preliminaries

The 3SUM problem

Problem (3SUM)

Given lists $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ of n reals, find (a, b, c) with $a \in \mathcal{A}, b \in \mathcal{B}, c \in \mathcal{C}$ such that $a + b = c$

Equivalently ask for $a + b + c = 0$

Forall $a \in \mathcal{A}$, $b \in \mathcal{B}$, $c \in \mathcal{C}$
    If $a + b = c$
        return (a, b, c)
return (nil, nil, nil)

3SUM in Fine-Grained Complexity

- The notion of 3SUM-hard problems
- 3SUM conjectures
  - hard: 3SUM cannot be solved in subquadratic time
  - weak: 3SUM cannot be solved in time $O(n^{2-\epsilon})$
- The hard conjecture has been recently refused:

2014 Gronlund and Pettie proposed the first subquadratic algorithm with time complexity $\mathcal{O}\left(n^2\left(\frac{\log\log n}{\log n}\right)^{2/3}\right)$

2017 Freund refined complexity to $\mathcal{O}\left(n^2\left(\frac{\log\log n}{\log n}\right)\right)$

2019 Chan gives a further improvement by about one more logarithmic factor $\mathcal{O}\left(n^2\left(\frac{(\log\log n)^{\mathcal{O}(1)}}{\log^2 n}\right)\right)$

3SUM and computational geometry

The 3SUM problem has a double correlation with computational geometry

- 3SUM has been used to prove many lower bounds of problems in computational geometry
- All the subquadratic 3SUM algorithms, at some point use a computational geometry problem
  - Gronlund,Pettie and Freund use the *dominance merge* problem
  - Chan uses *cuttings* in near-logarithmic dimensions

The linear time 2SUM algorithm

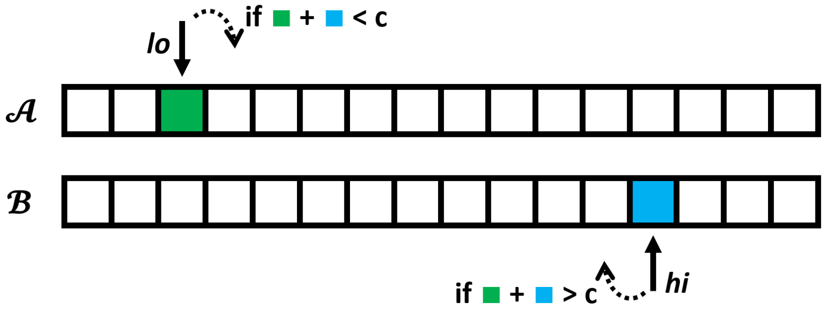
Problem (ORDERED-2SUM)

Given sorted lists $\mathcal{A}$, $\mathcal{B}$ of n reals, and a real c, find $a \in \mathcal{A}$ and $b \in \mathcal{B}$ such that $a + b = c$

The linear time 2SUM algorithm

Problem (ORDERED-2SUM)

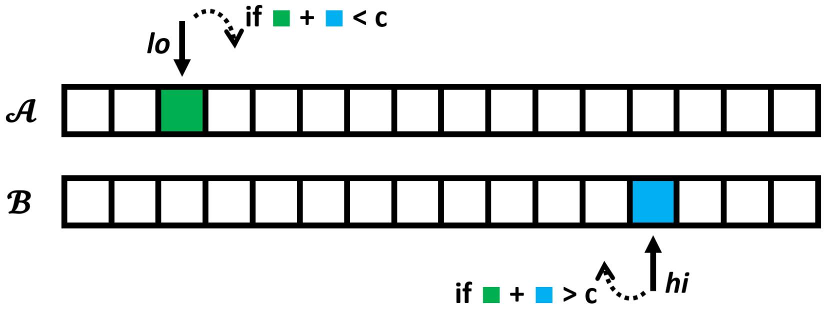
Given sorted lists $\mathcal{A}$, $\mathcal{B}$ of n reals, and a real c, find $a \in \mathcal{A}$ and $b \in \mathcal{B}$ such that $a + b = c$



The linear time 2SUM algorithm

Problem (ORDERED-2SUM)

Given sorted lists $\mathcal{A}$, $\mathcal{B}$ of n reals, and a real c, find $a \in \mathcal{A}$ and $b \in \mathcal{B}$ such that $a + b = c$



Invariant: elements $\mathcal{A}[i]$ with $i < lo$ and elements $\mathcal{B}[j]$ with $j > hi$ don't contribute to the solution

The quadratic time $3\textsc{sum}$ algorithm

Sort $\mathcal{A}$, sort $\mathcal{B}$

Forall $c \in \mathcal{C}$

    $(a, b, c) \leftarrow \textsc{ordered-2sum}(\mathcal{A}, \mathcal{B}, c)$

    If $(a, b, c) \neq (nil, nil, nil)$

        return (a, b, c)

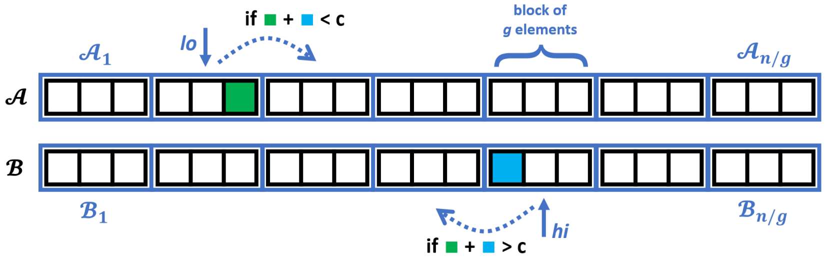
return (nil, nil, nil)

The Freund approach

A. Freund. "Improved subquadratic 3SUM". In: *Algorithmica* 77 (2017), pp. 440–458

Sketch (1)

1. Sort $\mathcal{A}$ and $\mathcal{B}$ and split them into n/g blocks of g elements
2. Preprocess $\mathcal{A}$ and $\mathcal{B}$
3. For each element $c \in \mathcal{C}$
   - 3a Initialize $lo \leftarrow 0$ and $hi \leftarrow \frac{n}{g}$
   - 3b Check if c is present in the sorted array $\mathcal{A}_{lo} + \mathcal{B}_{hi}$ of g^2 elements
   - 3c If such c does not exist, update lo and hi and go to 3b



Notation
$\mathcal{A}_i + \mathcal{B}_j = \{\mathcal{A}_i[h] + \mathcal{B}_j[k] : h, k \in [g]\}$

n/g blocks of g elements

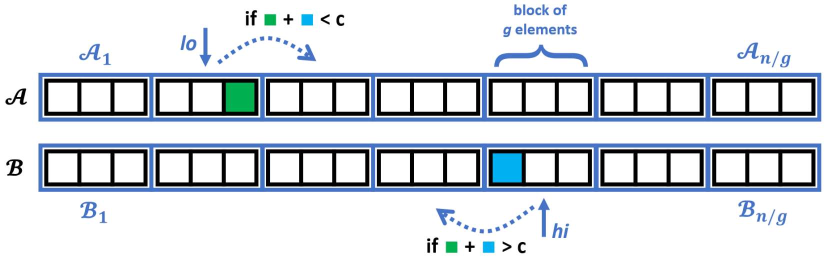
eprocess $\mathcal{A}$ and $\mathcal{B}$

3 For each element $c \in \mathcal{C}$

3a Initialize $lo \leftarrow 0$ and $hi \leftarrow \frac{n}{g}$

3b Check if c is present in the sorted array $\mathcal{A}_{lo} + \mathcal{B}_{hi}$ of g^2 elements

3c If such c does not exist, update lo and hi and go to 3b



Sketch (2)

- The preprocessing computes each sorted $\mathcal{A}_i + \mathcal{B}_j$
- The check for c in $\mathcal{A}_{lo} + \mathcal{B}_{hi}$ is a binary search
- The running time is $\mathcal{O}\left(n \log n + T_{\text{preprocessing}} + n^2 \frac{\log g}{g}\right)$

Sketch (2)

- The preprocessing computes each sorted $\mathcal{A}_i + \mathcal{B}_j$
- The check for c in $\mathcal{A}_{lo} + \mathcal{B}_{hi}$ is a binary search
- The running time is $\mathcal{O}\left(n \log n + T_{\text{preprocessing}} + n^2 \frac{\log g}{g}\right)$

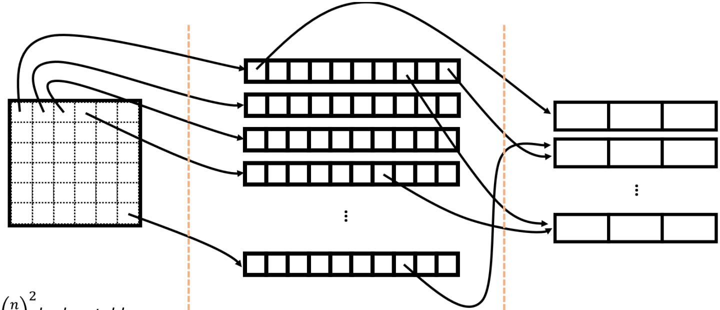
How to preprocess in subquadratic time?

- Cannot compute each ordered $\mathcal{A}_i + \mathcal{B}_j$ explicitly
- We will use a triple indirect data structure s.t.
    - We can access at $(\mathcal{A}_{lo} + \mathcal{B}_{hi})[k]$ in constant time
    - The preprocessing of the data structure is subquadratic
- The point in the triple indirection is to avoid the computation of shared elements in $\mathcal{A}_i + \mathcal{B}_j$

The triple indirection: idea

- Let A, B be two particular $\mathcal{A}_i$ and $\mathcal{B}_j$
- Define $\mathit{ranking}_{A,B}$ as the sequence of indices (i, j) where $i, j \in [g]$, sorted by increasing order of $A[i] + B[j]$
- Assumption: All the values in $A + B$ are different and so $\mathit{ranking}_{A,B}$ has a unique order
- Split $\mathit{ranking}_{A,B}$ into g^2/s chunks of size s
- To keep the preprocessing stage subquadratic, we want to enable **chunk sharing**
- We will compute chunks without explicitly constructing rankings

The triple indirection: access in constant time



The $\left(\frac{n}{g}\right)^2$ *lookup table*
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

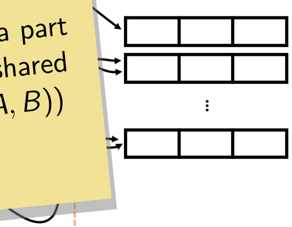
Each array represent a $ranking_{\{\mathcal{A}_i, \mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

ant time

Notation:

- $L[i,j]$ is the entry in the lookup table corresponding to blocks $\mathcal{A}_i$ and $\mathcal{B}_j$
- $RK_{A,B}$ is an ordered array (of size g^2/s) of pointers to chunks
- CA_T is a chunk array which stores a part (of size s) of $ranking_{A,B}$ (possibly shared for more than one pair of blocks (A,B))

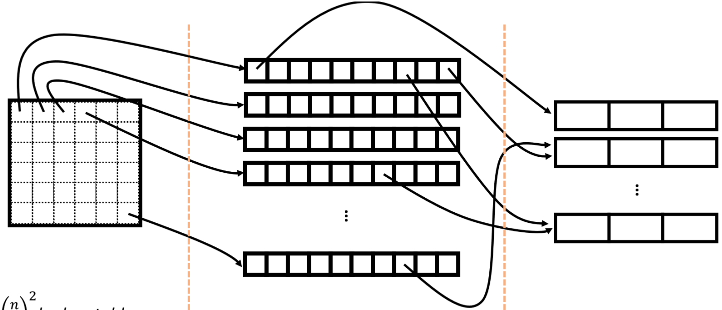


Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i,\mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

The triple indirection: access in constant time



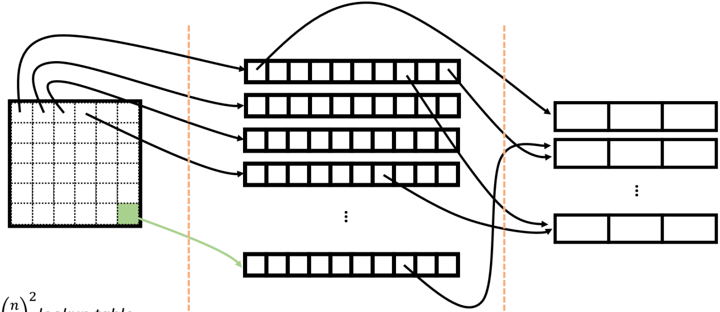
The $\left(\frac{n}{g}\right)^2$ *lookup table*
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i,\mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

$$(\mathcal{A}_i + \mathcal{B}_j)[k]$$

The triple indirection: access in constant time



The $\left(\frac{n}{g}\right)^2$ *lookup table*

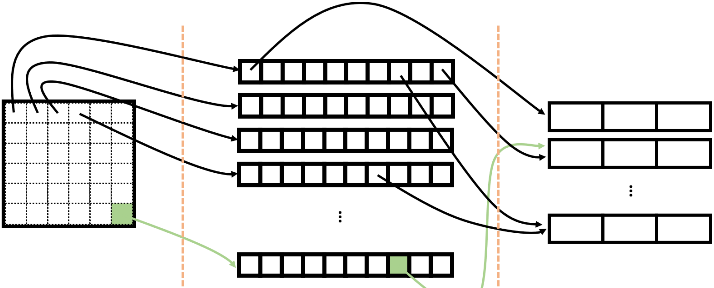
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i, \mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

$$(\mathcal{A}_i + \mathcal{B}_j)[k] \rightarrow L[i, j]$$

The triple indirection: access in constant time



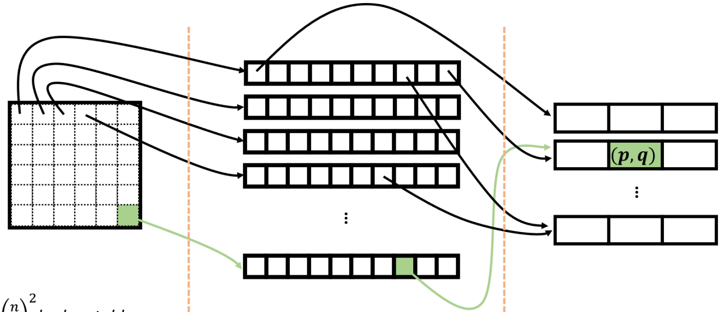
The $\left(\frac{n}{g}\right)^2$ *lookup table*
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i,\mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

$$(\mathcal{A}_i + \mathcal{B}_j)[k] \to L[i,j] \to RK_{\mathcal{A}_i,\mathcal{B}_j}[k/s]$$

The triple indirection: access in constant time



The $\left(\frac{n}{g}\right)^2$ *lookup table*

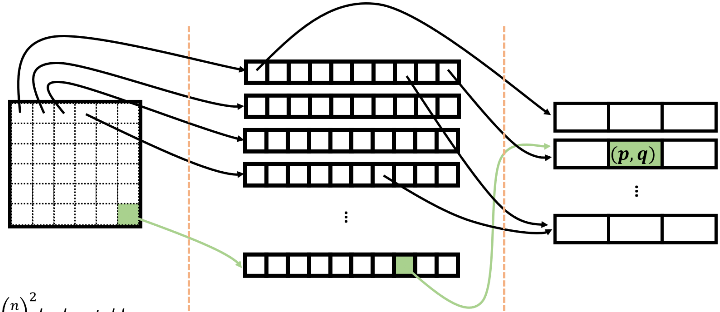
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i, \mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

$$(\mathcal{A}_i + \mathcal{B}_j)[k] \to L[i, j] \to RK_{\mathcal{A}_i, \mathcal{B}_j}[k/s] \to CA_T[((k-1) \bmod s) + 1]$$

The triple indirection: access in constant time



The $\left(\frac{n}{g}\right)^2$ *lookup table*
Each entry corresponds to $\mathcal{A}_i + \mathcal{B}_j$

Each array represent a $ranking_{\{\mathcal{A}_i, \mathcal{B}_j\}}$ and points to $\frac{g^2}{s}$ chunks

(possibly shared) *chunks* of size s

$$(\mathcal{A}_i + \mathcal{B}_j)[k] \to L[i,j] \to RK_{\mathcal{A}_i,\mathcal{B}_j}[k/s] \to CA_T[((k-1) \bmod s) + 1]$$

$$\to val_{\mathcal{A}_i,\mathcal{B}_j(p,q)} = \mathcal{A}_i[p] + \mathcal{B}_j[q]$$

Preprocessing time

Time $\mathcal{O}\left(\frac{n^2}{s}\right)$ to:

- allocate L
- point L's entries to RK arrays
- point RK's entries to chunk arrays

Preprocessing time

Time $\mathcal{O}\left(\frac{n^2}{s}\right)$ to:

- allocate L
- point L's entries to RK arrays
- point RK's entries to chunk arrays

How to compute (efficiently) chunk arrays?
Chunks should be small such that it is more convenient to enumerate all the outcomes rather than computing the content for each chunk

Computing chunks: exhaustive enumeration

Forall pairs A, B

Forall $e : 1 \leq e \leq g^2/s$

For every $S : |S| = s, S \subset [g]^2$

For every permutation π

    If S_π agrees with the e-th chunk of $ranking_{A,B}$

        $CA \leftarrow \text{NEW-CHUNK-ARRAY}(S_\pi)$

        $RK_{A,B}[e] = CA$

Computing chunks: exhaustive enumeration

Forall pairs A, B

Forall $e : 1 \leq e \leq g^2/s$

For every $S : |S| = s, S \subset [g]^2$

For every permutation π

  If S_π agrees with the e-th chunk of $ranking_{A,B}$

    $CA \leftarrow \textsc{new-chunk-array}(S_\pi)$

    $RK_{A,B}[e] = CA$

How to take advantage of chunk sharing?

$\rightarrow$ Reverse the order of operations

Computing chunks: exhaustive enumeration with chunk sharing

For every $S : |S| = s, S \subset [g]^2$
For every permutation π
Forall $e : 1 \leq e \leq g^2/s$
    $CA \leftarrow \textsc{new-chunk-array}(S_\pi)$
    Find all A, B : (S, π, e) agrees with the e-th chunk of $ranking_{A,B}$
    Forall such pairs A, B
        $RK_{A,B}[e] = CA$

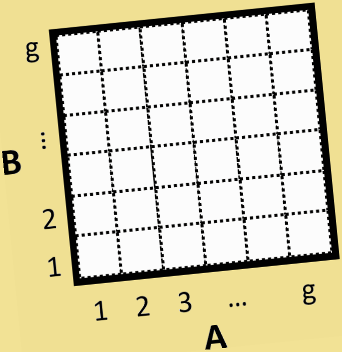
Computing chunks: exhaustive enumeration with chunk sharing

For every $S : |S| = s, S \subset [g]^2$
For every permutation π
Forall $e : 1 \leq e \leq g^2/s$
    $CA \leftarrow \textsc{new-chunk-array}(S_\pi)$
    Find all A, B : (S, π, e) agrees with the e-th chunk of $ranking_{A,B}$
    Forall such pairs A, B
        $RK_{A,B}[e] = CA$

- Most of the choices for (S, π, e) never agree with any $ranking_{A,B}$
  How to avoid excessive enumeration of (S, π, e)?

th chunk

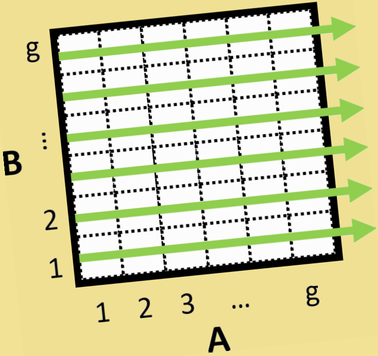
of $ranking_{A,B}$



$ranking_{A,B}$ can be represented by a $g \times g$ matrix

$king_{A,B}$

th chunk

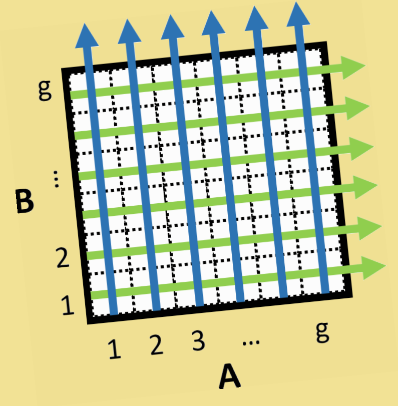


$ranking_{A,B}$ can be represented by a $g \times g$ matrix , but A is sorted, and so rows of the matrix are,

of $ranking_{A,B}$

king$_{A,B}$

...th chunk

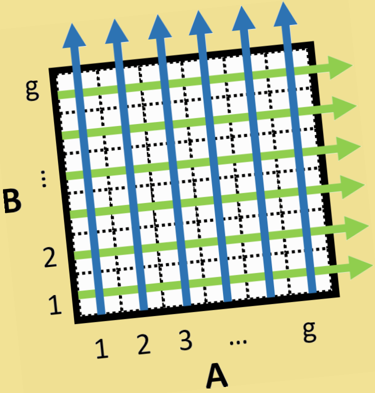


$ranking_{A,B}$ can be represented by a $g \times g$ matrix , but A is sorted, and so rows of the matrix are, and also B is sorted (and so columns of the matrix).

...of $ranking_{A,B}$

...$king_{A,B}$

th chunk

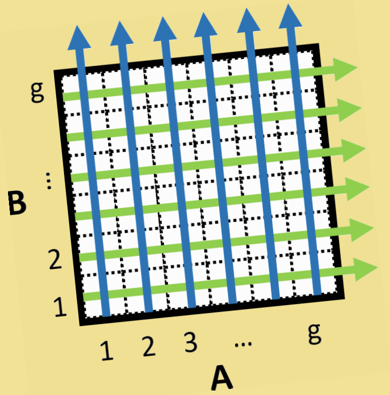


$ranking_{A,B}$ can be represented by a $g \times g$ matrix , but A is sorted, and so rows of the matrix are, and also B is sorted (and so columns of the matrix). As an example, regardless A and B, we will never find the pairs (g, g) and $(1, 1)$ in the same chunk.

of $ranking_{A,B}$

king$_{A,B}$

th chunk



of $ranking_{A,B}$

$king_{A,B}$

$ranking_{A,B}$ can be represented by a $g \times g$ matrix , but A is sorted, and so rows of the matrix are, and also B is sorted (and so columns of the matrix). As an example, regardless A and B, we will never find the pairs (g, g) and $(1, 1)$ in the same chunk. Furthermore, the pair (g, g) will never be in the first chunk.

Computing chunks: exhaustive enumeration with chunk sharing

For every $S : |S| = s, S \subset [g]^2$
For every permutation π
Forall $e : 1 \leq e \leq g^2/s$
    $CA \leftarrow \textsc{new-chunk-array}(S_\pi)$
    Find all A, B : (S, π, e) agrees with the e-th chunk of $ranking_{A,B}$
    Forall such pairs A, B
        $RK_{A,B}[e] = CA$

- Most of the choices for (S, π, e) never agree with any $ranking_{A,B}$
  How to avoid excessive enumeration of (S, π, e)?
- If the *find* procedure is just an enumeration, then the cost of chunk's computation will be greater than $\mathcal{O}(n^2)$
  How to avoid enumerating all pairs (A, B)?

Avoid exhaustive enumeration: idea

- As seen, a ranking for a fixed pair of blocks A, B can be represented by a $g \times g$ matrix $M_{A,B}$ where the value of cell (h, k) is $val(h, k) = A[h] + B[k]$. Rows and columns are sorted since A and B are.
- A path on M starting from the top left corner is a sequence of at most $2g - 1$ moves $\{Right, Down\}$ which ends falling off the right edge or bottom edge. If x is the last instruction, there are exactly g instructions labelled with x in the path.
- We will enumerate particular pairs of paths such that the set of squares in between them is a candidate set for a specific chunk
- There are $\left(\sum_{j=0}^{g-1} \frac{(j+g-1)!}{(g-1)!(j)!}\right)^2 \in \mathcal{O}(2^{4g})$ such pairs of paths

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

ontours

Recall
By assumption all values in $M_{A,B}$ are unique

efficiently determine the

$S_{A,B}$

- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the 2SUM algorithm (searching for $val(i,j)$)

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]
			A			

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i, j) we can efficiently determine the position r of (i, j) inside $ranking_{A,B}$
- r is the number of squares (h, k) such that $val(h, k) < val(i, j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i, j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2, 3, 4, 20, 30, 40\}$ and $B = \{9, 12, 15, 18, 21, 24\}$ and search for $(3, 3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm

Rules
Start from top left. If $val(h,k) \leq val(3,3)$ move right, otherwise move down. Continue until falling from edges.

ontours

n efficiently determine the

that $val(h,k) < val(i,j)$ and
the 2SUM algorithm (searching
$(i,j))$

B

g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the 2SUM algorithm

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = ()$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i, j) we can efficiently determine the position r of (i, j) inside $ranking_{A,B}$
- r is the number of squares (h, k) such that $val(h, k) < val(i, j)$ and can be computed in $\mathcal{O}(g)$ by a run of the 2SUM algorithm (searching for $val(i, j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2, 3, 4, 20, 30, 40\}$ and $B = \{9, 12, 15, 18, 21, 24\}$ and search for $(3, 3)$
- $contour^{3,3}_{A,B}$ is the path taken by the 2SUM algorithm
- $contour^{3,3}_{A,B} = (D)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D, D)$

Avoid exhaustive enumeration: contours

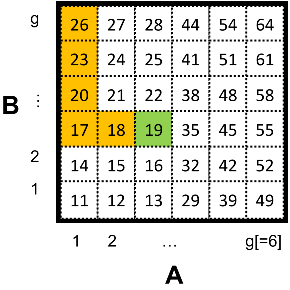
- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]
			A			

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D,D,D)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i, j) we can efficiently determine the position r of (i, j) inside $ranking_{A,B}$
- r is the number of squares (h, k) such that $val(h, k) < val(i, j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i, j)$)



- Consider $A = \{2, 3, 4, 20, 30, 40\}$ and $B = \{9, 12, 15, 18, 21, 24\}$ and search for $(3, 3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D, D, D, R)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

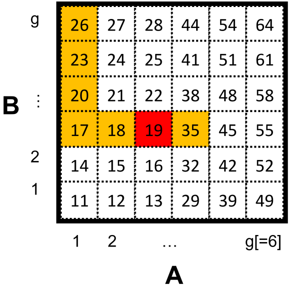
B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D,D,D,R,R)$

Avoid exhaustive enumeration: contours

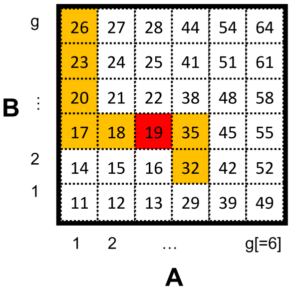
- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)



- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D,D,D,R,R,R)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)



- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D,D,D,R,R,R,D)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)

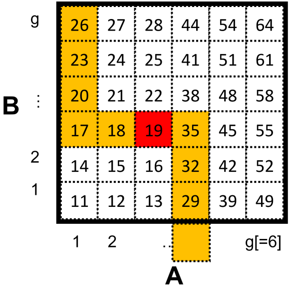
B						
g	26	27	28	44	54	64
	23	24	25	41	51	61
⋮	20	21	22	38	48	58
	17	18	19	35	45	55
2	14	15	16	32	42	52
1	11	12	13	29	39	49
	1	2	…			g[=6]

A

- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour_{A,B}^{3,3}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour_{A,B}^{3,3} = (D,D,D,R,R,R,D,D)$

Avoid exhaustive enumeration: contours

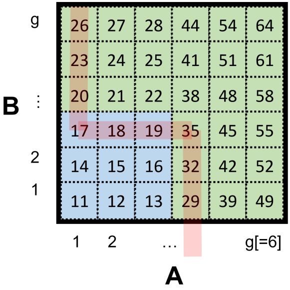
- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)



- Consider $A = \{2,3,4,20,30,40\}$ and $B = \{9,12,15,18,21,24\}$ and search for $(3,3)$
- $contour^{3,3}_{A,B}$ is the path taken by the $2\textsc{sum}$ algorithm
- $contour^{3,3}_{A,B} = (D,D,D,R,R,R,D,D,D)$

Avoid exhaustive enumeration: contours

- Given $M_{A,B}$ and a square (i,j) we can efficiently determine the position r of (i,j) inside $ranking_{A,B}$
- r is the number of squares (h,k) such that $val(h,k) < val(i,j)$ and can be computed in $\mathcal{O}(g)$ by a run of the $2\textsc{sum}$ algorithm (searching for $val(i,j)$)



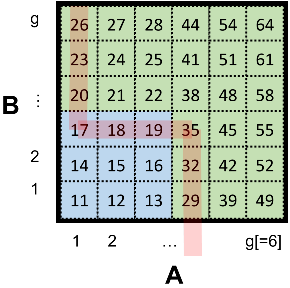
- A square (h,k) is said to be below $contour^{i,j}_{A,B}$ if there exists a $k' \geq k$ such that the path moves right from (h,k')
- In this case there are 9 squares below $contour^{3,3}_{A,B}$ and hence the pair $(3,3)$ is in position 9 of $ranking_{A,B}$

Rule
To calculate in $\mathcal{O}(g)$ the number of squares below $contour^{i,j}_{A,B}$, sum the indexes k of squares (h,k) from which the contour moves right

...ontours

...n efficiently determine the

... that $val(h,k) < val(i,j)$ and
... the 2SUM algorithm (searching
... $(i,j))$



- A square (h,k) is said to be below $contour^{i,j}_{A,B}$ if there exists a $k' \geq k$ such that the path moves right from (h,k')
- In this case there are 9 squares below $contour^{3,3}_{A,B}$ and hence the pair $(3,3)$ is in position 9 of $ranking_{A,B}$

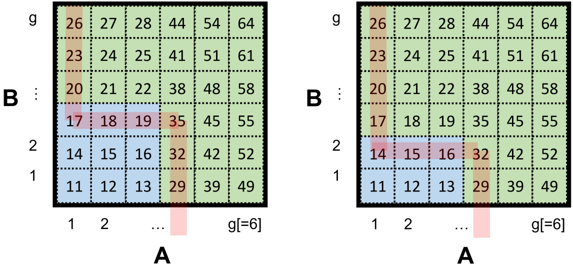
Avoid exhaustive enumeration: pairs of contours

- For any position $1 \leq r \leq g^2$ there is a unique square (i, j) such that there are exactly r squares below $contour^{i,j}_{A,B}$

- The set S is the e-th chunk of $ranking_{A,B}$
  if and only if
  S is the set of squares above $contour^{k,l}_{A,B}$ and below $contour^{k',l'}_{A,B}$ where (k, l) (resp. (k', l')) is such that below $contour^{k,l}_{A,B}$ (resp. $contour^{k',l'}_{A,B}$) there are exactly $(e-1)s$ (resp. es) squares

- P' dominates P if every square above P' is also above P

Avoid exhaustive enumeration: pairs of contours



- $contour^{3,3}_{A,B}$ dominates $contour^{3,2}_{A,B}$
- Consider chunks of dimension 3
- Under $contour^{3,3}_{A,B}$ there are $3 \times 3 = 9$ squares
- Under $contour^{3,2}_{A,B}$ there are $2 \times 3 = 6$ squares
- The set $\{(1,3),(2,3),(3,3)\}$ is the third chunk of $ranking_{A,B}$

Computing chunks: refined enumeration

```
Forall pairs of paths P, P' s.t. P' dominates P
Check if between P and P' there are exactly s squares
Check if ∃e s.t. below P there are (e − 1)s squares
Let S be the set of squares between P and P'
Forall (k, l), (k', l') possible anchors of P and P'
Forall permutations π such that (k', l') is the last element
    CA ← NEW-CHUNK-ARRAY(S_π)
    Find all pairs A, B : (S, π, P, P', (k, l), (k', l')) agrees with the
e-th chunk of ranking_{A,B}
    Forall such pairs A, B
        RK_{A,B}[e] = CA
```

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

We need to verify three properties

1 P is $contour^{k,l}_{A,B}$

2 P' is $contour^{k',l'}_{A,B}$

3 S_π is a sequence of increasing order of values in $A + B$

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

We need to verify three properties

1. P is $contour^{k,l}_{A,B}$
   - $|P| - 1$ inequalities: one for each visited point in the contour
   - $A[i] + B[j] < A[k] + B[l]$ if P moves right from (i, j)
   - $A[k] + B[l] < A[i] + B[j]$ otherwise
2. P' is $contour^{k',l'}_{A,B}$
3. S_π is a sequence of increasing order of values in $A + B$

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

We need to verify three properties

1. P is $contour_{A,B}^{k,l}$
   - $|P| - 1$ inequalities: one for each visited point in the contour
   - $A[i] + B[j] < A[k] + B[l]$ if P moves right from (i, j)
   - $A[k] + B[l] < A[i] + B[j]$ otherwise
2. P' is $contour_{A,B}^{k',l'}$
   - translation to $|P'| - 1$ inequalities as point 1
3. S_π is a sequence of increasing order of values in $A + B$

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

We need to verify three properties

1. P is $contour_{A,B}^{k,l}$
   - $|P| - 1$ inequalities: one for each visited point in the contour
   - $A[i] + B[j] < A[k] + B[l]$ if P moves right from (i, j)
   - $A[k] + B[l] < A[i] + B[j]$ otherwise
2. P' is $contour_{A,B}^{k',l'}$
   - translation to $|P'| - 1$ inequalities as point 1
3. S_π is a sequence of increasing order of values in $A + B$
   - translation to $s - 1$ inequalities
   - denote by (a_k, b_k) the pair $S_\pi[k]$
   - $A[a_k] + B[b_k] < A[a_{k+1}] + B[b_{k+1}]$ forall $1 \leq r < s$

Find pairs A, B that agree with $(S, \pi, P, P', (k,l), (k',l'))$

We need to verify three properties

1. P is $contour^{k,l}_{A,B}$
   - $|P| - 1$ inequalities: one for each visited point in the contour
   - $B[j] - B[l] < A[k] - A[i]$ if P moves right from (i,j)
   - $B[l] - B[j] < A[i] - A[k]$ otherwise
2. P' is $contour^{k',l'}_{A,B}$
   - translation to $|P'| - 1$ inequalities as point 1
3. S_π is a sequence of increasing order of values in $A + B$
   - translation to $s - 1$ inequalities
   - denote by (a_k, b_k) the pair $S_\pi[k]$
   - $B[b_k] - B[b_{k+1}] < A[a_{k+1}] - A[a_k]$ forall $1 \leq r < s$

Last step is called Fredman's trick and is also used in the Chan approach

$n\ (S, \pi, P, P', (k, l), (k', l'))$

one for each visited point in the contour

- $B[j] - B[l] < A[k] - A[i]$ if P moves right from (i, j)
- $B[l] - B[j] < A[i] - A[k]$ otherwise

2. P' is $contour^{k',l'}_{A,B}$
   - translation to $|P'| - 1$ inequalities as point 1
3. S_π is a sequence of increasing order of values in $A + B$
   - translation to $s - 1$ inequalities
   - denote by (a_k, b_k) the pair $S_\pi[k]$
   - $B[b_k] - B[b_{k+1}] < A[a_{k+1}] - A[a_k]$ forall $1 \leq r < s$

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

- To check the condition for a particular A, B we just need to check if $v^B < v^A$

Notation
v^B (resp. v^A) is a k-dimensional vector ($k = |P| + |P'| - 3 + s < 4g + s$) such that the u-th component is the lhs (resp. rhs) of the u-th inequality described

$\pi, P, P', (k, l), (k', l')$)

, B we just need to check if

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

- To check the condition for a particular A, B we just need to check if $v^B < v^A$
- Fixed $(S, \pi, P, P', (k, l), (k', l'))$, we want to find all the pairs (A, B) such that $v^B < v^A$

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

- To check the condition for a particular A, B we just need to check if $v^B < v^A$
- Fixed $(S, \pi, P, P', (k, l), (k', l'))$, we want to find all the pairs (A, B) such that $v^B < v^A$
- We could enumerate all the n^2/g^2 pairs but that's too much time

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

- To check the condition for a particular A, B we just need to check if $v^B < v^A$
- Fixed $(S, \pi, P, P', (k, l), (k', l'))$, we want to find all the pairs (A, B) such that $v^B < v^A$
- We could enumerate all the n^2/g^2 pairs but that's too much time
- Note that values of v^A depend only on A (the same for v^B and B) and so there is a total of $2n/g$ vectors to construct

Find pairs A, B that agree with $(S, \pi, P, P', (k, l), (k', l'))$

- To check the condition for a particular A, B we just need to check if $v^B < v^A$
- Fixed $(S, \pi, P, P', (k, l), (k', l'))$, we want to find all the pairs (A, B) such that $v^B < v^A$
- We could enumerate all the n^2/g^2 pairs but that's too much time
- Note that values of v^A depend only on A (the same for v^B and B) and so there is a total of $2n/g$ vectors to construct
- we can now reformulate the problem as an instance of *dominance merge* problem which admits an efficient solution

Problem (ALL-AGREEING-PAIRS)

Given n/g vectors v^A and n/g vectors v^B find all pairs A, B such that $v^B < v^A$

The dominance merge problem

Problem (DOMINANCE-MERGE)

Given N vectors in $\mathbb{R}^d$ such that each vector v has a color $c(v) \in \{1, 2\}$, find every pair (a, b) such that $c(a) = 1$, $c(b) = 2$ and $a < b$

The dominance merge problem

Problem (DOMINANCE-MERGE)

Given N vectors in $\mathbb{R}^d$ such that each vector v has a color $c(v) \in \{1, 2\}$, find every pair (a, b) such that $c(a) = 1$, $c(b) = 2$ and $a < b$

- Can be solved recursively in $\mathcal{O}(|D| + 2^{2d} N^{1,5})$ where D is the answer set

The dominance merge problem

Problem (DOMINANCE-MERGE)

Given N vectors in $\mathbb{R}^d$ such that each vector v has a color $c(v) \in \{1, 2\}$, find every pair (a, b) such that $c(a) = 1$, $c(b) = 2$ and $a < b$

- Can be solved recursively in $\mathcal{O}(|D| + 2^{2d} N^{1,5})$ where D is the answer set
- In our case we have $2n/g$ vectors (color 2 to those of type v^A and color 1 otherwise) and less than $4g + s$ dimensions

The dominance merge problem

Problem (DOMINANCE-MERGE)

Given N vectors in $\mathbb{R}^d$ such that each vector v has a color $c(v) \in \{1, 2\}$, find every pair (a, b) such that $c(a) = 1$, $c(b) = 2$ and $a < b$

- Can be solved recursively in $\mathcal{O}(|D| + 2^{2d} N^{1,5})$ where D is the answer set
- In our case we have $2n/g$ vectors (color 2 to those of type v^A and color 1 otherwise) and less than $4g + s$ dimensions
- We can solve ALL-AGREEING-PAIRS problem in time $\mathcal{O}\left(|D| + 2^{8g+2s}\left(\frac{2n}{g}\right)^{1,5}\right)$

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2
- Number of permutations of S bounded by $s! \leq 2^{s \log s}$

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2
- Number of permutations of S bounded by $s! \leq 2^{s \log s}$
- Time to solve a ALL-AGREEING-PAIRS problem:
  $\mathcal{O}\left(|D| + 2^{8g+2s}\left(\frac{2n}{g}\right)^{1,5}\right)$

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2
- Number of permutations of S bounded by $s! \leq 2^{s \log s}$
- Time to solve a ALL-AGREEING-PAIRS problem: $\mathcal{O}\left(|D| + 2^{8g+2s}\left(\frac{2n}{g}\right)^{1,5}\right)$
- $\tilde{D}$ the total number of dominance pairs through all invocations of the dominance algorithm

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2
- Number of permutations of S bounded by $s! \leq 2^{s \log s}$
- Time to solve a ALL-AGREEING-PAIRS problem: $\mathcal{O}\left(|D| + 2^{8g+2s}\left(\frac{2n}{g}\right)^{1,5}\right)$
- $\tilde{D}$ the total number of dominance pairs through all invocations of the dominance algorithm

The total time spent to compute chunks is

$$\mathcal{O}\left(\tilde{D} + g^2 2^{12g+2s+s\log s}\left(\frac{2n}{g}\right)^{1,5}\right)$$

Recall

Dominance pairs are in 1:1 correspondence with entries of arrays $RK_{A,B}$

So we can remove the term $\tilde{D}$ since we have already considered it

g^2

$s! \leq 2^{s \log s}$

blem:

- $\tilde{D}$ the total number of dominance pairs through all invocations of the dominance algorithm

The total time spent to compute chunks is

$$\mathcal{O}\left(\tilde{D} + g^2 2^{12g+2s+s\log s}\left(\frac{2n}{g}\right)^{1,5}\right)$$

Computing chuncks: complexity

- Number of paths (P, P') bounded by 2^{4g}
- Number of anchors in (P, P') bounded by g^2
- Number of permutations of S bounded by $s! \leq 2^{s \log s}$
- Time to solve a ALL-AGREEING-PAIRS problem:
  $\mathcal{O}\left(|D| + 2^{8g+2s}\left(\frac{2n}{g}\right)^{1,5}\right)$
- $\tilde{D}$ the total number of dominance pairs through all invocations of the dominance algorithm

The total time spent to compute chunks is $\mathcal{O}\left(g^2 2^{12g+2s+s\log s}\left(\frac{2n}{g}\right)^{1,5}\right)$

Relaxing the value uniqueness hypothesis

pothesis

Recall the assumption

All the elements in $A+B$ are distinct (and so all the values for pairs in $ranking_{A,B}$ are distinct)

Relaxing the value uniqueness hypothesis

The correctness of the preprocessing is guaranteed by the following properties of $ranking_{A,B}$:

…pothesis

…aranteed by the following

Notation
Denote by $(i, j) \prec (h, k)$ the fact that the pair (i, j) occurs before the pair (h, k) in the sequence $ranking_{A,B}$

Relaxing the value uniqueness hypothesis

The correctness of the preprocessing is guaranteed by the following properties of $ranking_{A,B}$:

1. The order of values respects the order of index pairs
   $val(i,j) < val(h,k) \implies (i,j) \prec (h,q)$

Relaxing the value uniqueness hypothesis

The correctness of the preprocessing is guaranteed by the following properties of $ranking_{A,B}$:

1. The order of values respects the order of index pairs
   $val(i,j) < val(h,k) \implies (i,j) \prec (h,q)$
2. The order of values is monotone in the vertical and horizontal grid directions
   - $i < h \implies (i,j) \prec (h,j)$
   - $j < k \implies (i,j) \prec (i,k)$

Relaxing the value uniqueness hypothesis

The correctness of the preprocessing is guaranteed by the following properties of $ranking_{A,B}$:

1. The order of values respects the order of index pairs
   $val(i,j) < val(h,k) \implies (i,j) \prec (h,q)$
2. The order of values is monotone in the vertical and horizontal grid directions
   - $i < h \implies (i,j) \prec (h,j)$
   - $j < k \implies (i,j) \prec (i,k)$
3. Given pairs (i,j) and (h,k), it is *easy* to decide the relation $(i,j) \prec (h,k)$ and the comparison can be decomposed into an A part and a B part

Importance of properties
1) allows binary search 2) allows the partitioning by countors and 3) allows the use of the *dominance merge* algorithm

...pothesis

...aranteed by the following

... of index pairs

... $val(h,k) \implies (i,j) \prec (h,q)$

2. The order of values is monotone in the vertical and horizontal grid directions
   - $i < h \implies (i,j) \prec (h,j)$
   - $j < k \implies (i,j) \prec (i,k)$
3. Given pairs (i,j) and (h,k), it is *easy* to decide the relation $(i,j) \prec (h,k)$ and the comparison can be decomposed into an A part and a B part

Relaxing the value uniqueness hypothesis

Solution 1: redefine the order of appearance such that there are no equalities and the three properties are satisfied

$$
\begin{aligned}
(i,j) \prec (h,k) \iff & (val(i,j) < val(h,k)) \vee \\
& (val(i,j) = val(h,k) \wedge i < h) \vee \\
& (val(i,j) = val(h,k) \wedge i = h \wedge j < k)
\end{aligned}
$$

Relaxing the value uniqueness hypothesis

Solution 1: redefine the order of appearance such that there are no equalities and the three properties are satisfied

$$\begin{aligned}(i,j) \prec (h,k) \iff & (val(i,j) < val(h,k)) \vee \\ & (val(i,j) = val(h,k) \wedge i < h) \vee \\ & (val(i,j) = val(h,k) \wedge i = h \wedge j < k)\end{aligned}$$

Solution 2: map elements $val(i,j)$ into a totally ordered universe (with addition and subtraction) such that squares' value is unique, and redefine the algorithm in terms of this objects. For example:

$$val_{A,B}(i,j) = A[i] + B[j] \longrightarrow (A[i] + B[j], i, j)$$

- pointwise addition and subtraction
- lexicographical order

$\rightarrow$ the three properties are satisfied

Subquadratic 3SUM: overall complexity

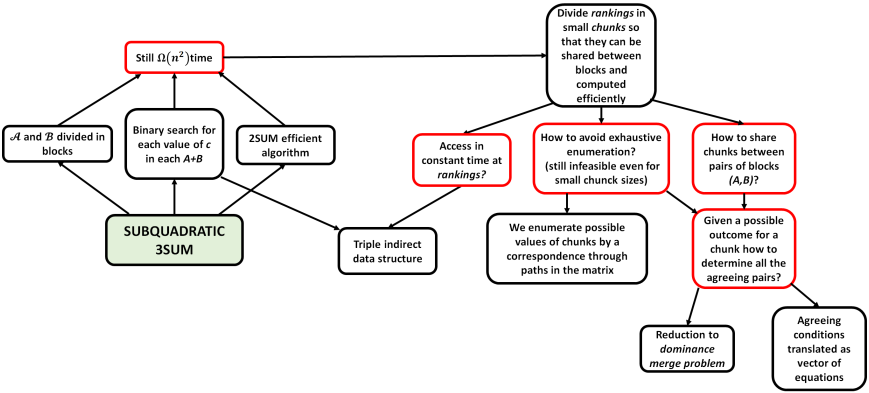
1. Time for the binary searches $\mathcal{O}\left(n^2 \frac{\log g}{g}\right)$
2. Time to prepare the data structure $\mathcal{O}\left(\frac{n^2}{s}\right)$
3. Time to compute chunks $\mathcal{O}\left(g^2 2^{12g+2s+s\log s}\left(\frac{2n}{g}\right)^{1,5}\right)$

If we put $s = \Theta(\frac{g}{\log g})$ and $g = \frac{1}{31}\log n$ we finally obtain the desired complexity of $\mathcal{O}\left(n^2 \frac{\log\log n}{\log n}\right)$

Final notes on chunks' computation

- $\mathcal{A} + \mathcal{B}$ has n^2 elements
- The exhaustive enumeration computes $\binom{g^2}{s} g^2 s!$ possible chunks
- The refined enumeration computes only $\mathcal{O}(2^{4g} g^2 s!)$ possible chunks which is $\mathcal{O}((\log^2 n) n^{5/31})$ plugging our choices for g and s
- Since each chunk has size $s = \Theta(\frac{g}{\log g}) = \Theta(\frac{\log n}{\log \log n})$ elements, in total we compute only $\mathcal{O}\left(\frac{\log n}{\log \log n}(\log^2 n) n^{5/31}\right) = \mathcal{O}\left(\frac{(\log^3 n) n^{5/31}}{\log \log n}\right)$ elements instead of n^2

Final ingredients



Conclusions

Concluding remarks

- It has been shown that there are subquadratic algorithms for 3SUM
- It is still unknown if the weak 3SUM conjecture holds (even thought it is widely believed, recall that the same held for the hard conjecture)
- The preprocessing is the key to reach a subquadratic result
- It seems that the preprocessing can't to do anything special rather than use a series of tricks
- If the weak conjecture is false, it seems to me that we are far away from obtaining efficient algorithms since we are still not able to build a data structure which exploits properties of the problem
- Even thought there exist subquadratic solutions, I suspect that the overhead introduced by structure's complexity of subquadratic algorithms makes more practical to use just a simple quadratic solution with low constants

Bibliography I

[CH20] T. M. Chan and Q. He. "Reducing 3SUM to convolution-3SUM". In: *Symposium on Simplicity in Algorithms*. SIAM. 2020, pp. 1–7.

[Cha08] T. M. Chan. "All-pairs shortest paths with real weights in O (n 3/log n) time". In: *Algorithmica* 50 (2008), pp. 236–243.

[Cha19] T. M. Chan. "More logarithmic-factor speedups for 3SUM,(median,+)-convolution, and some geometric 3SUM-hard problems". In: *ACM Transactions on Algorithms (TALG)* 16.1 (2019), pp. 1–23.

[Fre17] A. Freund. "Improved subquadratic 3SUM". In: *Algorithmica* 77 (2017), pp. 440–458.

Bibliography II

[GO95] A. Gajentaan and M. H. Overmars. "On a class of O (n2) problems in computational geometry". In: *Computational geometry* 5.3 (1995), pp. 165–185.

[GP14] A. Grønlund and S. Pettie. "Threesomes, degenerates, and love triangles". In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE. 2014, pp. 621–630.

[Pat10] M. Patrascu. "Towards polynomial lower bounds for dynamic problems". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 603–610.

[PS12] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

Bibliography III

[Wil18] V. V. Williams. "On some fine-grained questions in algorithms and complexity". In: *Proceedings of the international congress of mathematicians: Rio de janeiro 2018*. World Scientific. 2018, pp. 3447–3487.