CSEP 521
Applied Algorithms
Spring 2005

Research Projects
of
Richard Ladner

Outline for Tonight

- Reduction of subset sum optimization to subset sum decision.
- Windows scheduling for periodic jobs.
- Student Evaluations
- Cache efficient dynamic programming
- Tactile graphics

Windows Scheduling

with

Amotz Bar-Noy

Tami Tamir

Windows Scheduling Problem Definition

- n unit length repetitive jobs with positive integer windows $w_1, w_2, \ldots, w_n$
- m processors.
- Scheduling goal: assign jobs on processors so that
  - Job i is scheduled on some processor at least once every w_i time slots.
  - No two jobs are scheduled in the same time slot on the same processor.
- Optimization goal: Given $w_1, w_2, \ldots, w_n$ minimize the number of processors.

Applications

- Video broadcast scheduling
  - On one channel a video can be broadcast so that the worst case waiting time is strictly smaller than the video length
- Periodic maintainance
  - Maintainance must be do at least so often
- Push systems
  - Ads, sports scores, DJ average must be displayed at least so often

Basic Lower Bound

- Let W = $w_1, w_2, \ldots, w_n$ and define

$$h(W) = \sum_{i=1}^{n} \frac{1}{w_i}$$

- Theorem: $\lceil h(W) \rceil$ is a lower bound on the number of number of processors needed to schedule W.
- Proof: Job i requires $1/w_i$ of a processor

Example 1

- W = 1,2,3 and m = 2

P_1:	1	1	1	1	1	1	…
P_2:	2	3	2	3	2	3	…

- This is a perfect schedule, that is, each job i is scheduled every w'_i slots for $w'_i \leq w_i$
- 3 is scheduled more often than its window requirement

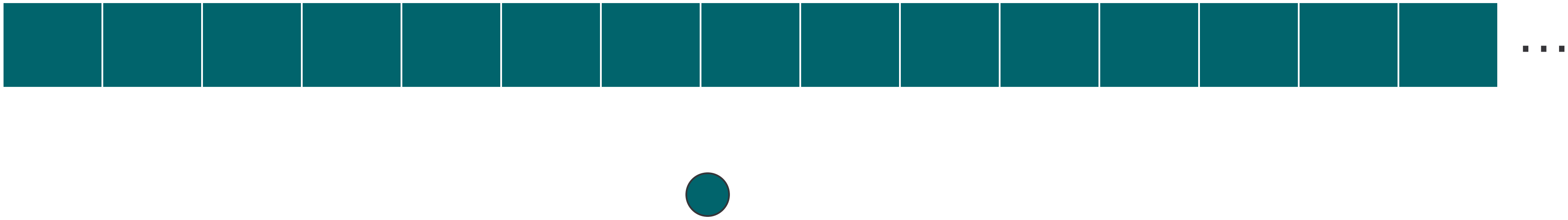
Example 2

- W = 4,5,6,7,8 and m = 1

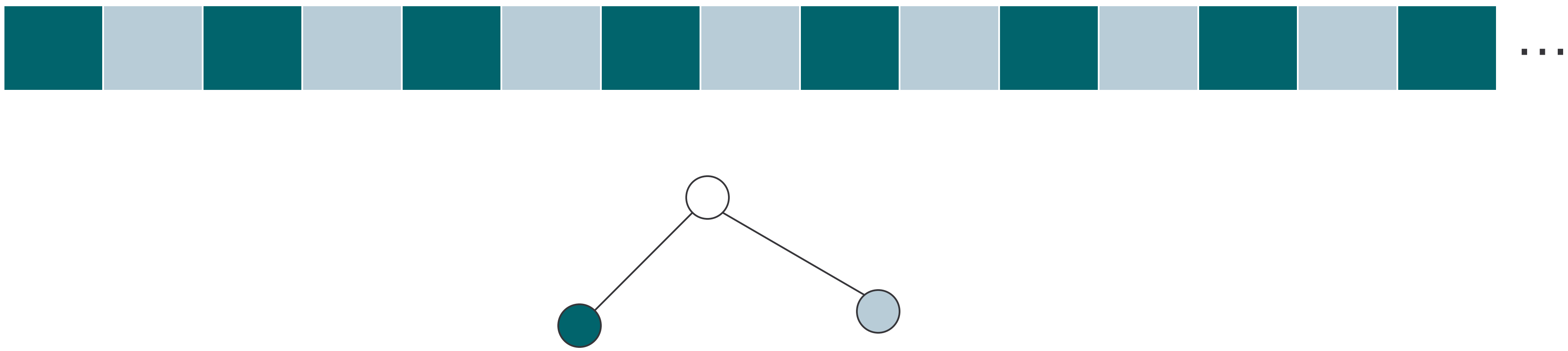
P_1:

4	6	5	7	4	8	5	6	4	7	5	8	4	6	5	…

Tree Representation of Perfect Schedules

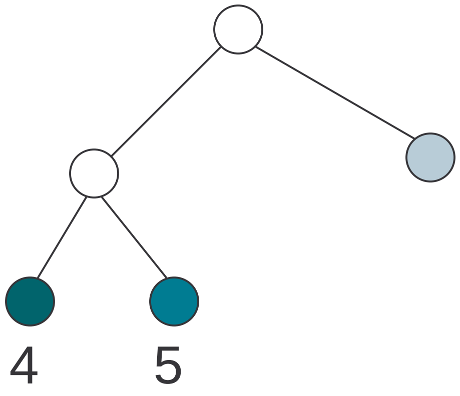


Tree Representation of Perfect Schedules



Tree Representation of Perfect Schedules

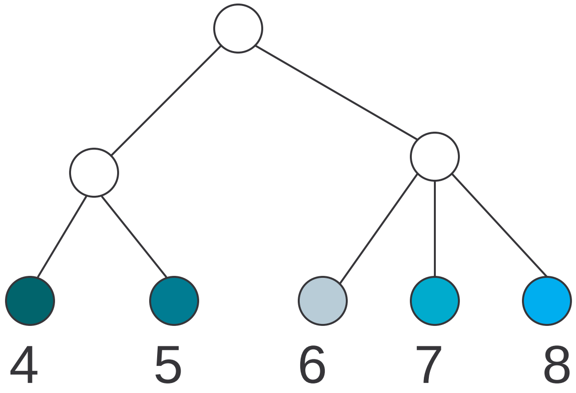
4		5		4		5		4		5		4		5	…



Each node is scheduled periodically every p times where p is the product of degrees of its ancesters.

Tree Representation of Perfect Schedules

4	6	5	7	4	8	5	6	4	7	5	8	4	6	5	…



The Rest of the Talk

- Buffer scheme
- Approximation
- Video-on-demand

Are Perfects Schedules Sufficient?

- No!
- W = 3,5,8,8,8 and m = 1
- There is no perfect schedule on one processor
- Non-perfect windows schedule

3	5	8a	3	8b	5	3	8c	8a	3	5	8b	3	8c	5	3	8a	8b	3	5	8c	…

- Non-perfect schedules can be found using a search technique call the buffer scheme.

Impossibility of Perfect 3,5,8,8,8

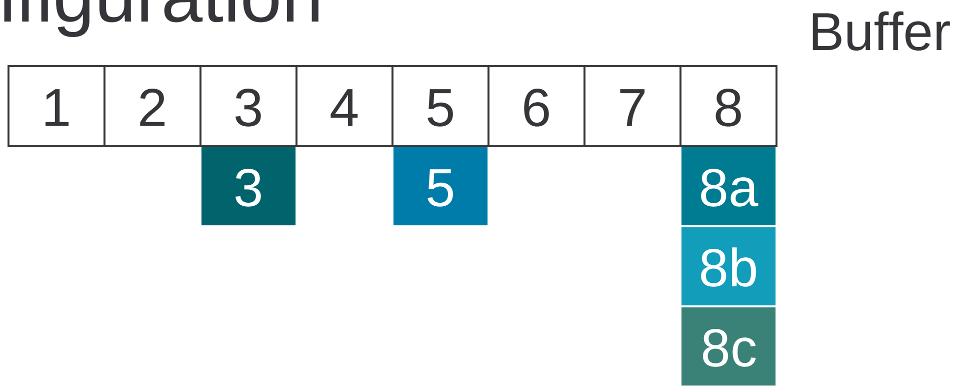
- 3 must have period 1,2, or 3
- But, $1/2 + 1/5 + 3/8 > 1$
- Hence 3 has period 3.
- 5 must have period 1,2,3,4, or 5
- But, $1/3 + 1/3 + 3/8 > 1$
- 5 must have period 4 or 5.
- But, gcd(4,3) = 1 and gcd(5,3) = 1, Chinese remainder theorem implies there must be a slot in common to the schedules of both 4 and 3, and 5 and 3. $\Rightarrow\Leftarrow$

Buffer Scheme

- A technique for searching all possible schedules.
- Can find non-perfect schedules.
- Can be used to prove impossibility.
- By adding deterministic rules it can be used as an on-line scheduler (jobs can inserted and deleted at each slot).

Buffer Scheme by Example

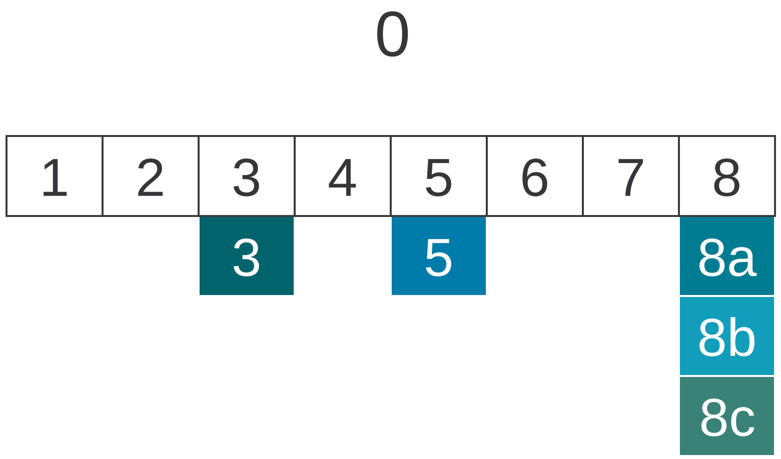
- If job i is in buffer location j, then i must be scheduled within j slots.
- Non-deterministic
- Complete - every schedule can be modeled
- Initial configuration



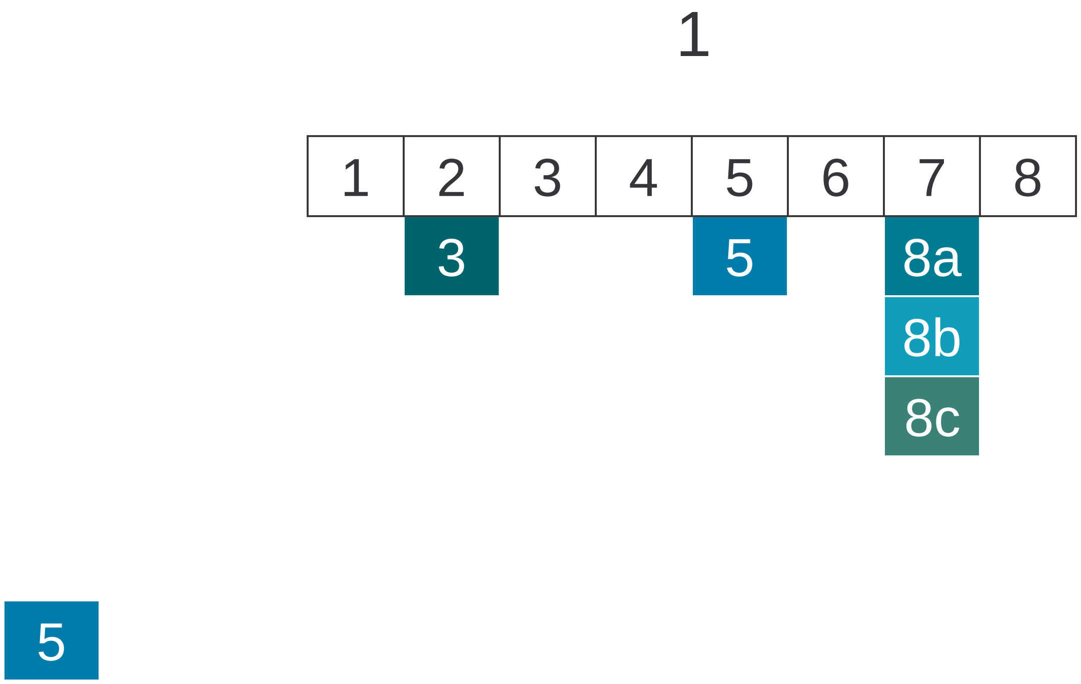
Buffer

1	2	3	4	5	6	7	8
		3		5			8a
							8b
							8c

Buffer Scheme Example



Buffer Scheme Example

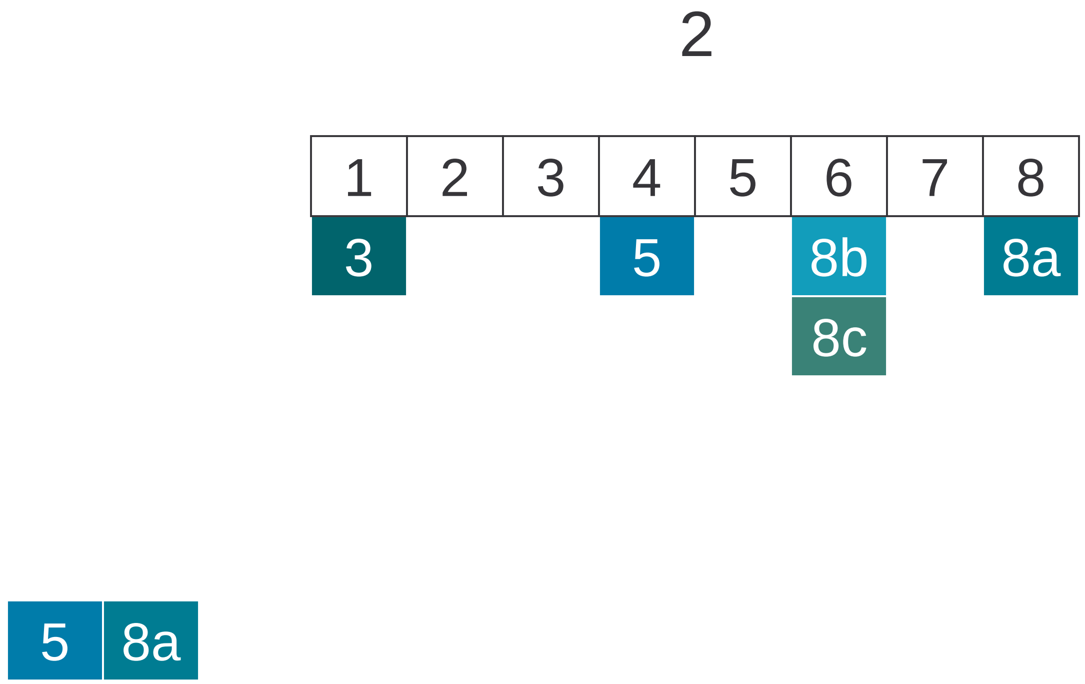


1

1	2	3	4	5	6	7	8
	3			5		8a	
						8b	
						8c	

5

Buffer Scheme Example

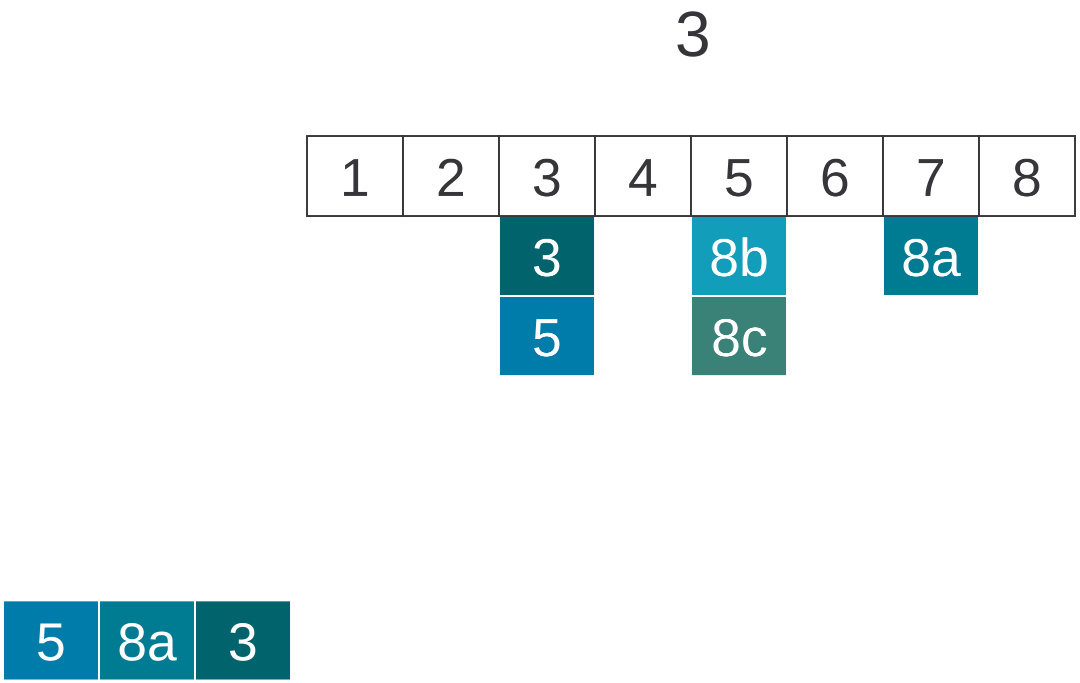


2

1	2	3	4	5	6	7	8
3			5		8b		8a
					8c		

5	8a

Buffer Scheme Example

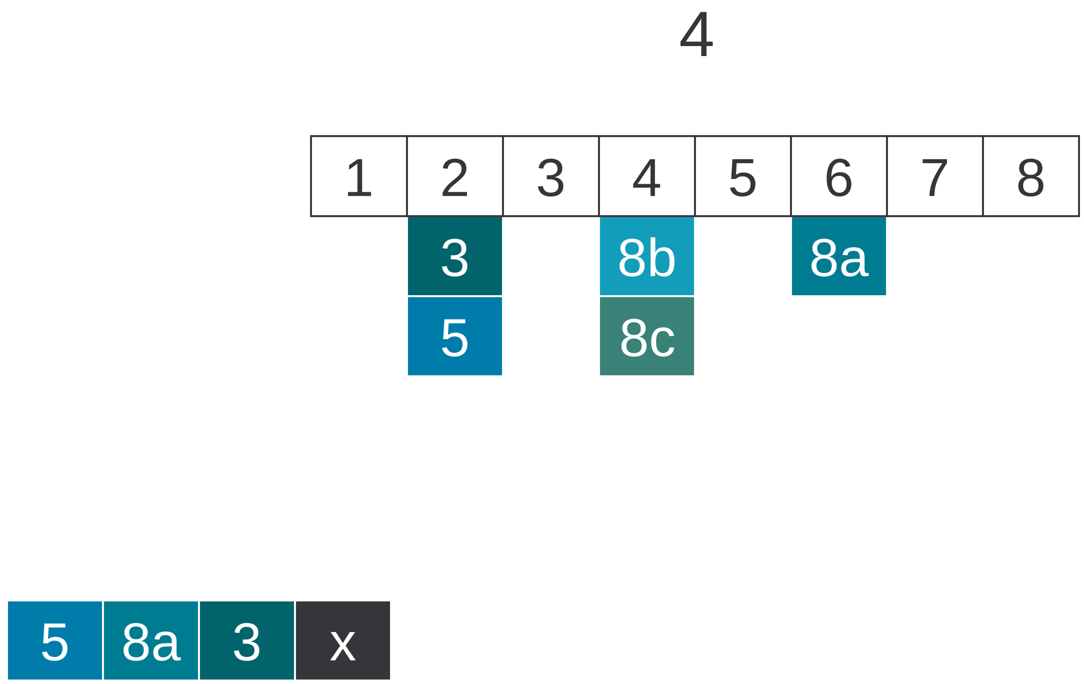


3

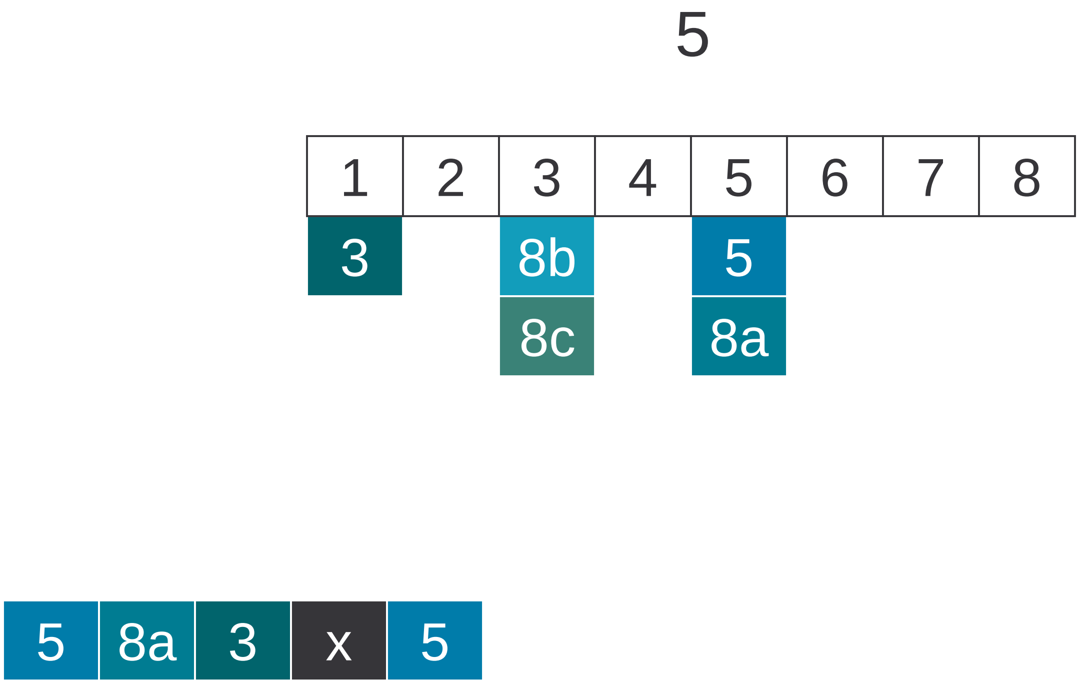
1	2	3	4	5	6	7	8
		3		8b		8a	
		5		8c			

5	8a	3

Buffer Scheme Example



Buffer Scheme Example



5

1	2	3	4	5	6	7	8
3		8b		5			
		8c		8a			

5	8a	3	x	5

Buffer Scheme Example

6

1	2	3	4	5	6	7	8
	8b	3	5				
	8c		8a				

5	8a	3	x	5	3

Buffer Scheme Properties

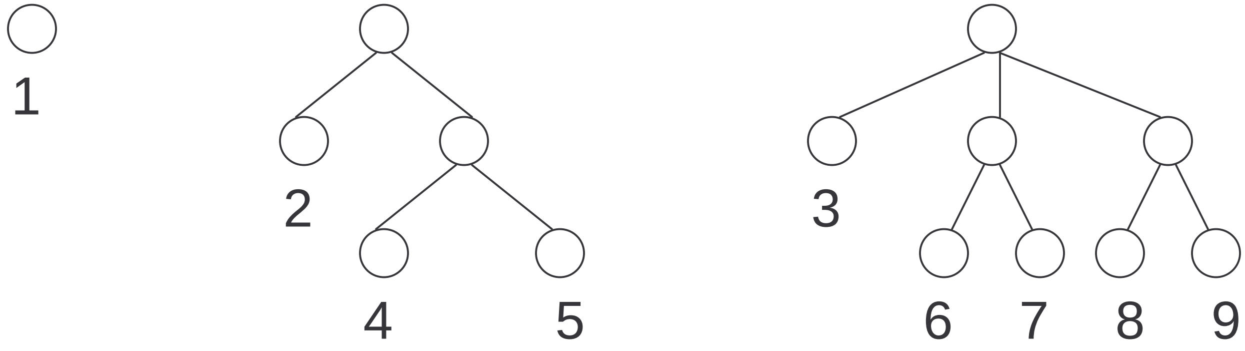
- Generalize to multiple processors.
  - Schedule $\leq$ m at each slot.
- Non-deterministic finite state machine.
- A cycle reachable from the start state is a schedule.
- Exhaustive search leads to impossibility.
  - Requires early dead-end detection for speed
- By adding a priority scheme it can be made into an on-line windows scheduling algorithm.

Buffer Scheme Impossibility

- W_{10} = 1,2,3,4,5,6,7,8,9,10 and m = 3

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{10}\approx 2.929$$

- Exhausive search proves there is no windows schedule for W_{10} on 3 processors.
- There is a perfect schedule for W_9



Approximate Windows Scheduling

- Given W, define H(W) to be the minimum number of processors needed to schedule W.
- Theorem: $H(W) = h(W) + O(\log(h(W)))$.
- The schedule can be found in polynomial time.
- Corollary: As $h(W) \to \infty$, windows scheduling is polynomial time approximable with approximation ratio 1.

Special Case: Powers of Two

- Special Case: if all w_i are powers of two then

$$H(W) = \lceil h(W) \rceil = \left\lceil \sum_{i=1}^{n} \frac{1}{w_i} \right\rceil$$

- Example: W = 2,2,2,4,8,16

2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

2	4	2	8	2	4	2	16	2	4	2	8	2	4	2	

…

Upper Bound by Rounding

$$H(W) \le \lceil 2h(W) \rceil$$

- Proof: Schedule page i every 2^k slots where 2^k is the largest power of two $\le w_i$. (Rounding)
- Example: W = 3,3,4,12

  Use W’ = 2,2,4,8   for scheduling

3	3	3	3	3	3	3	3	⋯
4	12			4				⋯

Asymptotic Upper Bound

$$H(W) \leq h(W) + e \cdot \ln(h(W)) + \eta$$

where $h(W) = \sum_{i=1}^{n} \frac{1}{w_i} > 1$ $\qquad e \approx 2.71828$

$\eta \approx 7.3595$

Main Ideas in the Upper Bound

- Expand Special Case: If all w_i are of the form $u2^k$ for a fixed u, then

$$H(W) = \lceil h(W) \rceil$$

- Multiple Rounding
- Schedule some windows to fill processors and schedule the residual recursively
- Do all this “optimally”

Multiple Rounding

W = 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

Round using 3,4,5

W_3 = 3,6,7,12,13,14,15
W'_3 = 3,6,6,12,12,12,12

W_4 = 2,4,8,9,16,17,18,19
W'_4 = 2,4,8,8,16,16,16,16

W_5 = 5,10,11
W'_5 = 5,10,10

Find Residual

W = 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

Round using 3,4,5

W_3 = 3,6,7,12,13,14,15
W'_3 = 3,6,6,12,12,12,12

1 processor

W_4 = 2,4,8,9,16,17,18,19
W'_4 = 2,4,8,8,16,16,16,16

W_5 = 5,10,11
W'_5 = 5,10,10

Find Residual

W = 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

Round using 3,4,5

W_3 = 3,6,7,12,13,14,15
W'_3 = 3,6,6,12,12,12,12

1 processor

W_4 = 2,4,8,9,16,17,18,19
W'_4 = 2,4,8,8,16,16,16,16

1 processor + 16,17,18,19

W_5 = 5,10,11
W'_5 = 5,10,10

Find Residual

W = 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

Round using 3,4,5

W_3 = 3,6,7,12,13,14,15
W'_3 = 3,6,6,12,12,12,12

1 processor

W_4 = 2,4,8,9,16,17,18,19
W'_4 = 2,4,8,8,16,16,16,16

1 processor + 16,17,18,19

W_5 = 5,10,11
W'_5 = 5,10,10

0 processors + 5,10,11

Solve Residual

W^r = 5,10,11,16,17,18,19

Round using 2

W_2 = 5,10,11,16,17,18,19
W'_2 = 4, 8, 8, 16,16,16,16

1 processor

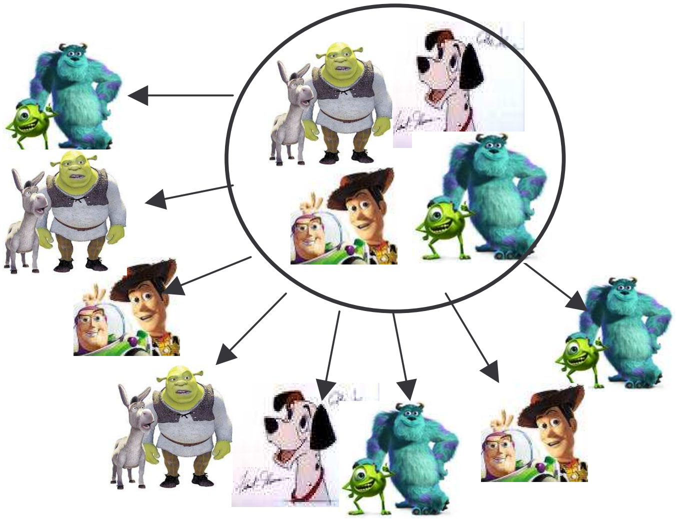
Total is 3 processors, while simple rounding needs 4.

Summary Asymptotic Bounds

$$\lceil h(W) \rceil \leq H(W) \leq h(W) + e \cdot \ln(h(W)) + \eta$$

- Upper bound is polynomial time
- Rounding sets optimized for the construction

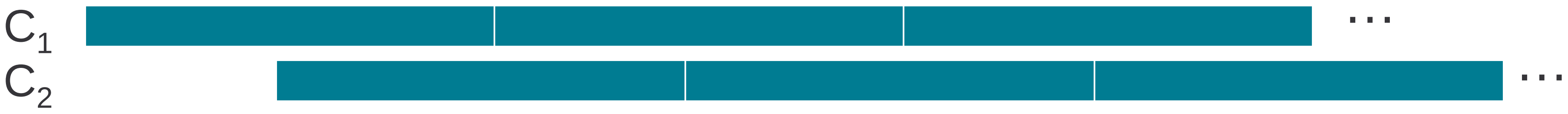
Video-on-Demand Systems



- A database of media objects (movies).
- A limited number of channels.
- Movies are broadcast based on customer demand
- The goal: Minimizing clients' maximum waiting time (delay).
- Broadcasting schemes: For popular movies, the system does not wait for client requests, but broadcasts these movies continuously.

Background

- Staggered broadcasting, [Dan, Sitaram, Shahabuddin, 96]:

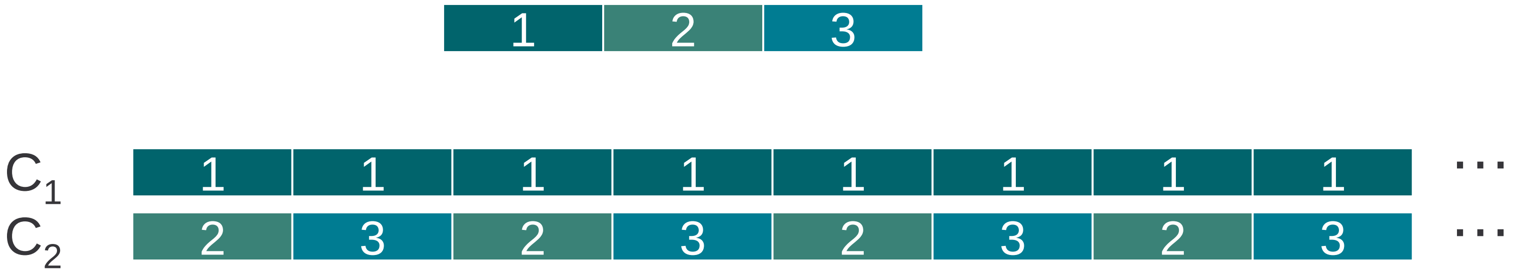


Delay = 1/2

Note: each channel is at the playback bandwidth.

Backgound

- Pyramid Broadcasting, [Viswanathan, Imielinski, 96]:
  - Partition the movie into segments. Early segments are transmitted more frequently.
  - As segments are received either playback or buffer for future playback



1	2	3

C_1	1	1	1	1	1	1	1	1	⋯
C_2	2	3	2	3	2	3	2	3	⋯

Delay = 1/3

Note that W = 1,2,3 and m = 2

Windows Sceduling for Video

- Shifting, [Bar-noy, Ladner, Tamir, 2003], Polyharmonic Broadcasting [Paris, 1999]
- Video divided into s equal size segments.
- For a constant $d > 0$ define
  $W = d, d+1, d+2, \ d+s-1$, that is, $w_i = d+i-1$
- Theorem: A windows schedule for W on m processors is a video schedule with s segments on m channels with delay d/s.

Example

- W = 4,5,6,7,8 and m = 1 (c.f. Example 2)

Windows Schedule

4	6	5	7	4	8	5	6	4	7	5	8	4	6	5	…

Video Schedule

1	3	2	4	1	5	2	3	1	4	2	5	1	3	2	…

Delay = 4/5

Lower Bounds

- Lowerbound [Engebretsen, Sudan, 2002], [Gao,Kurose,Towsley, 2002], [Hu, 2001] Delay for m channels is bounded below by

$$\frac{1}{e^m - 1}$$

m	1	2	3	4	5	6
$1/(e^m-1)$	.582	.157	.052	.019	.007	.002
1 hour video	35 min	9.5 min	3 min	1 min	25 sec	7 sec

Asymptotic Upper Bound

- For every m and $\varepsilon > 0$, there is a video schedule that achives delay

$$(1+\varepsilon)\frac{1}{e^{m}-1}$$

Limiting the Number of Segments

- Given s segments what is the best delay possible on one channel. Use the buffer scheme!

segments	range	delay
5	4..8	0.800
6	5..10	0.883
7	5..11	0.714
8	6..13	0.750
120*	75..194	0.625

0.582 optimal

* Uses RR^2 algorithm, not buffer scheme

Additional Work

- Receive k channels out of m for video scheduling. [Evans, Kirkpatrick, 2004]
- On-line windows scheduling
  - Buffer scheme - insertions and deletions
  - $H(W) + O(H(W)^{1/2})$ - insertions only
- Windows scheduling with lengths
  - Roughly a 2-approximation algorithm

Cache Efficient Dynamic Programming

with

Cary Cherng

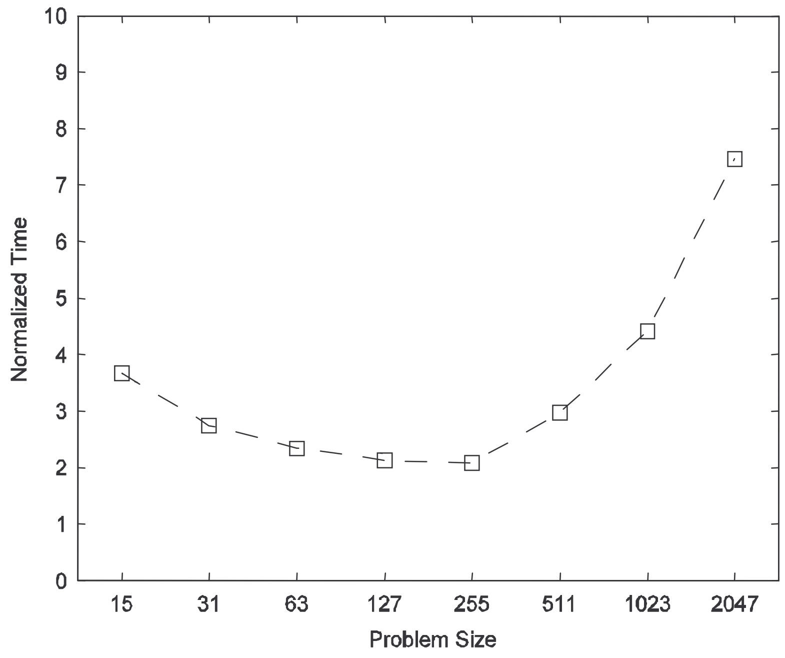
Problem Statement

- Given $x_1, x_2, \ldots, x_n$ in nonassociative semiring
  - Multiplication is nonassociative
  - Additive inverses not required
- Find the sum of all ways $x_1x_2\ldots x_n$ can be completely parenthesized
  - $n = 4$

    $x_1((x_2x_3)x_4) + x_1(x_2(x_3x_4)) + (x_1x_2)(x_3x_4) + ((x_1x_2)x_3)x_4 + (x_1(x_2x_3))x_4$
- Examples of nonassociative semirings:
  - Matrix Chain Product
  - Context-free Language Recognition

Standard Dynamic Programming Solution - CYK

- Runs in $O(n^3)$
- Poor cache behavior for large n
- Normalized Time = Time/n^3



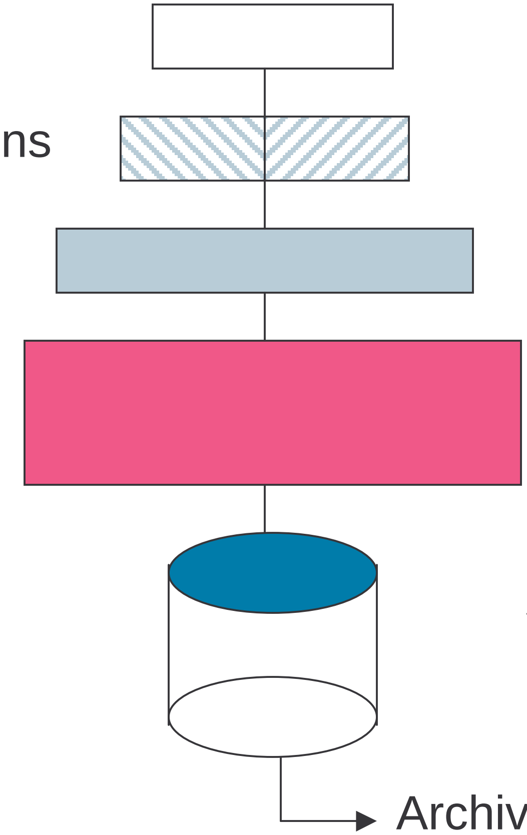
Outline

- The Cache
- Dynamic Programming
  - Standard Dynamic Programming Solution - CYK (Cocke, Younger, Kasami 1965)
  - Valiant’s Algorithm (1975)
- Experiments
  - Timing comparison
  - Cache misses

The Cache

- Standard dynamic programming has poor cache locality
- Divide and conquer algorithms typically have good cache locality

Memory Hierarchy



64-128 ALU registers

SRAM; a few ns

On-chip cache: split I-cache; D-cache 8-128KB

SRAM/DRAM;
≈ 10-20 ns

Off-chip cache; 128KB - 4MB

DRAM; 40-100 ns

Main memory; up to 10GB

a few milliseconds

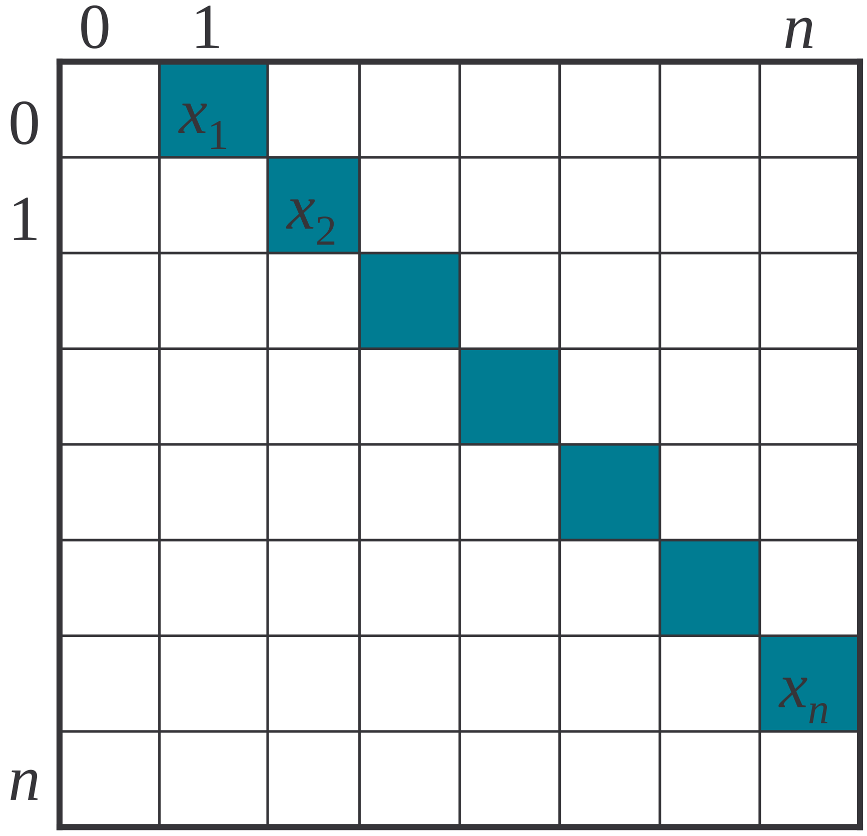
Secondary memory; many GB

Archival storage

Standard Dynamic Programming

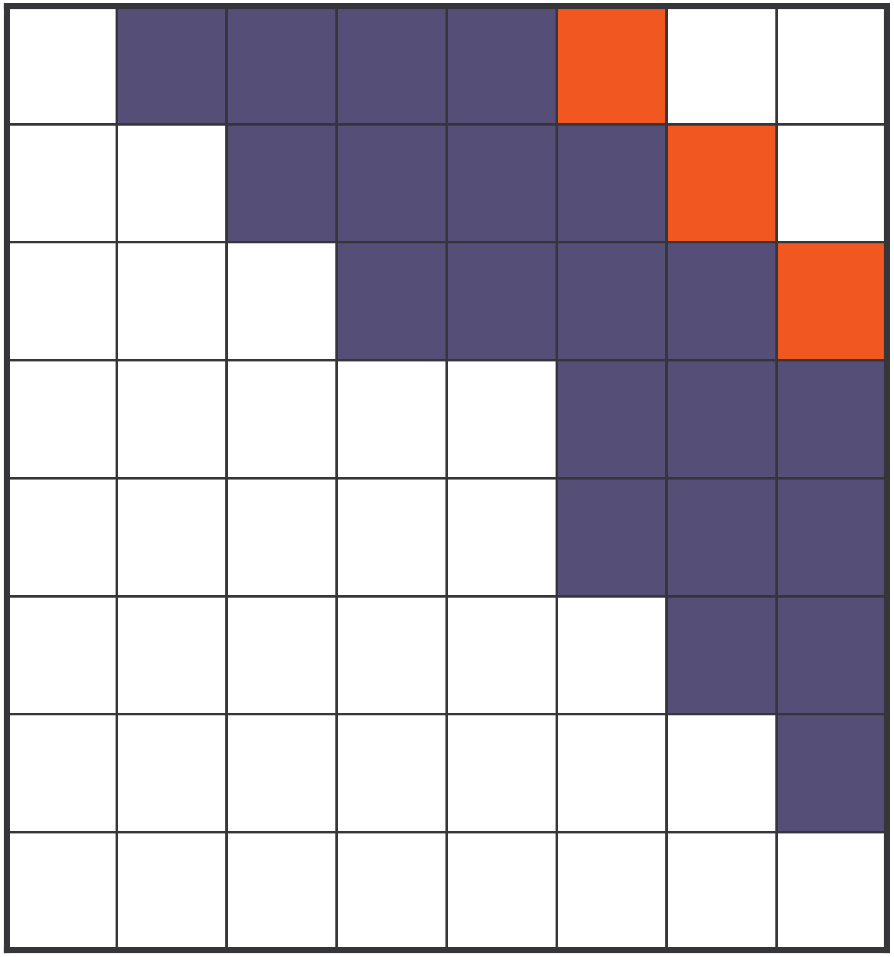
- Matrix of size $n+1$
- Input off the diagonal
- For i = 1 to n

```
For $i$ = 1 to $n$
  $D_{i-1,i} = x_i$
For $j$ = 2 to $n$
  For $i = j$ - 2 to 0
    For $k = i$ + 1 to $j$ - 1
      $D_{ij} = D_{ij} + D_{ik} D_{kj}$
```



Dynamic Programming

- Computing D_{ij} requires accessing blue region
- Many cache misses if the matrix is large due to poor locality



Valiant's Algorithm

- Valiant proved in 1975 that context-free language recognition could be done in $O(\mathrm{n}^{2.81})$ using Strassen's matrix multiplication algorithm [1969]
- Current fastest Boolean matrix multiplication runs in $O(\mathrm{n}^{2.376})$ due to Coppersmith and Winograd [1987;1990]
- Implies $O(\mathrm{n}^{2.376})$ for context-free language recognition using Valiant's Algorithm

Valiant’s Algorithm

- Divide and Conquer giving good cache locality
- Reorganizes the computation of CKY
- The algorithm can be applied to nonassociative semirings

Matrix Multiply and Accumulate

- Basic building block for the new algorithm.
- U, W, and Z are square arrays of power of 2 size.

$$U := U + W \cdot Z$$

Blocked Matrix Multiply and Accumulate

- Reduces the number of cache misses
- Recursive

$$\begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} := \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} + \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$$

- 8 matrix multiplies
- U_{11} is computed as

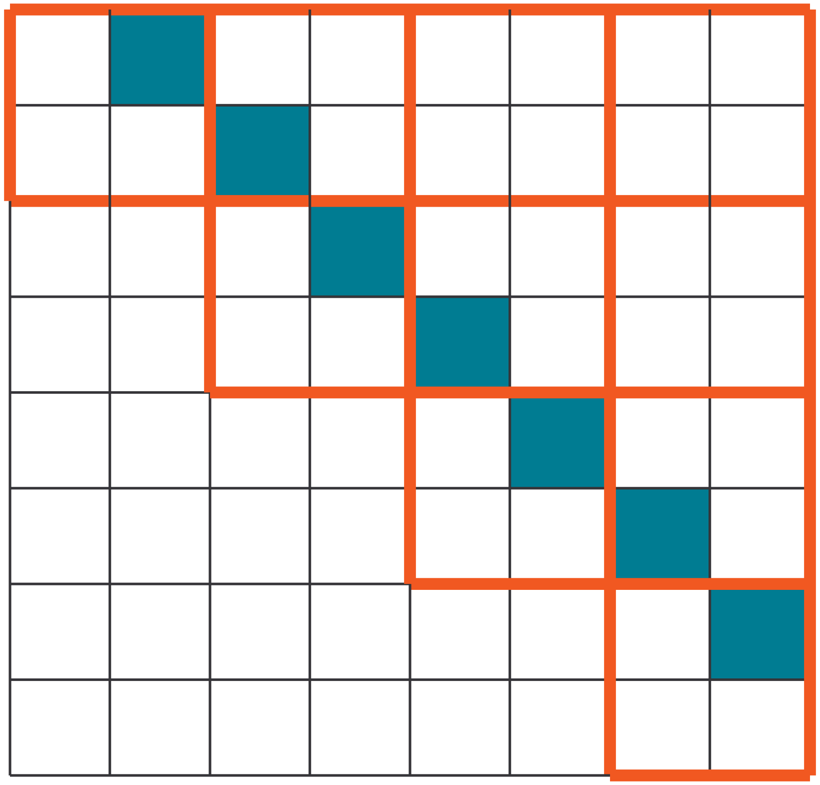
  $U_{11} := U_{11} + W_{11} Z_{11}$

  $U_{11} := U_{11} + W_{12} Z_{21}$

Valiant's Algorithm

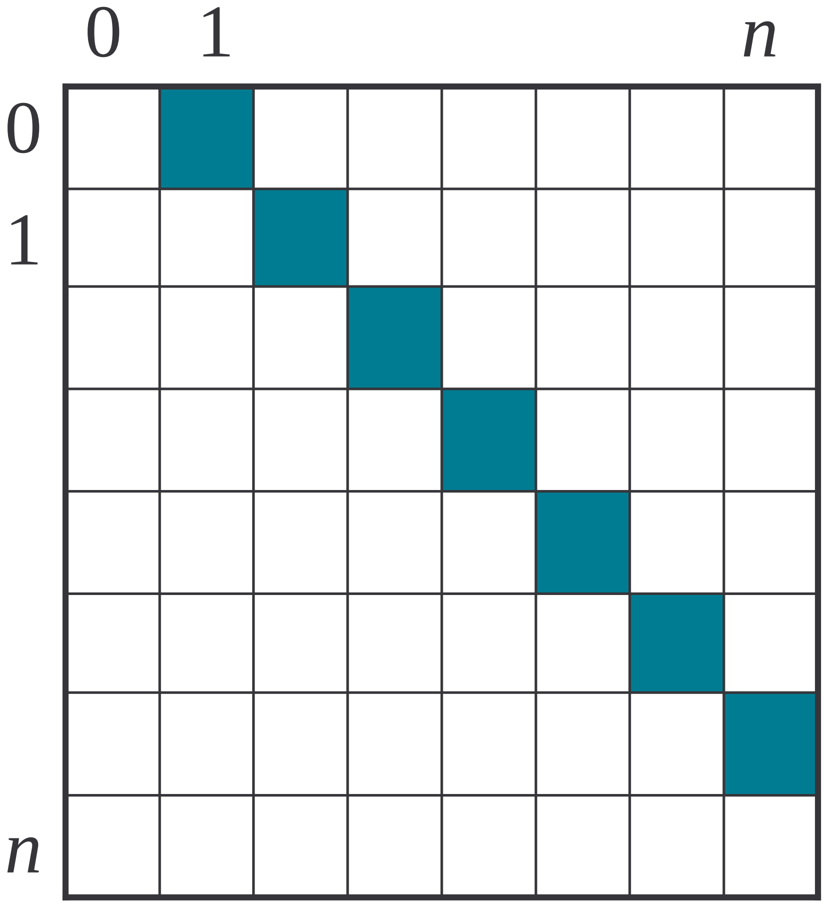
- View X as blocked

$$X = \begin{bmatrix} X_{11} & X_{12} & & \\ & X_{22} & X_{23} & \\ & & X_{33} & X_{34} \\ & & & X_{44} \end{bmatrix}$$



Valiant’s Algorithm

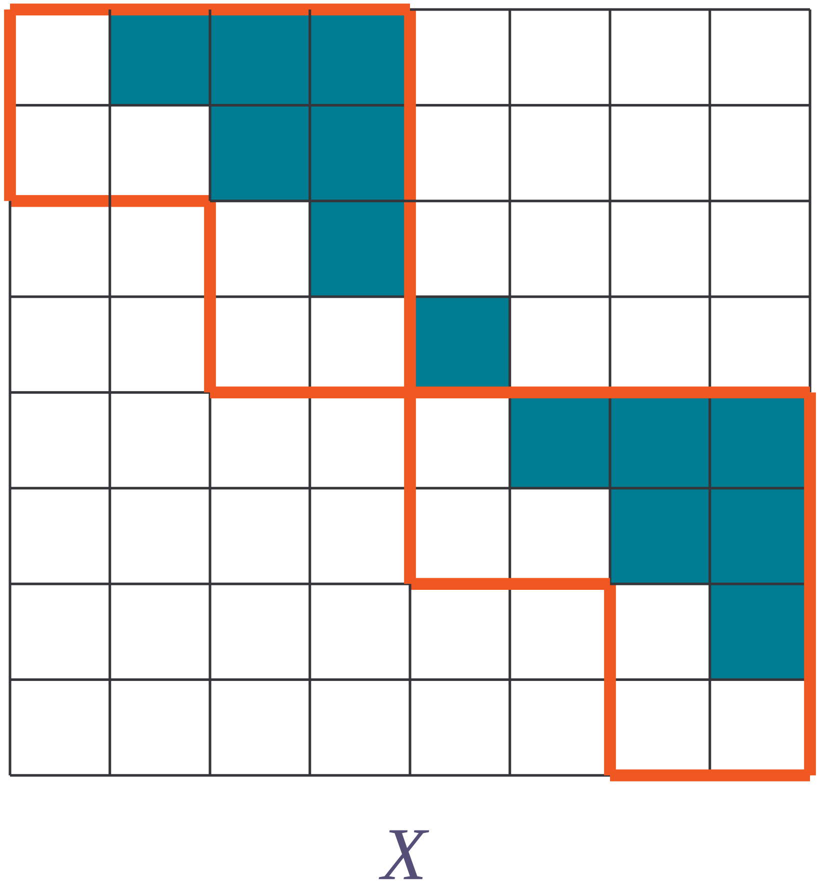
- Directly applies only when $n = 2^k - 1$
- Matrix X of size $n+1$
- X^+ means the result of applying Valiant’s algorithm to X



Star Function Input

- A helper function
- Precondition:

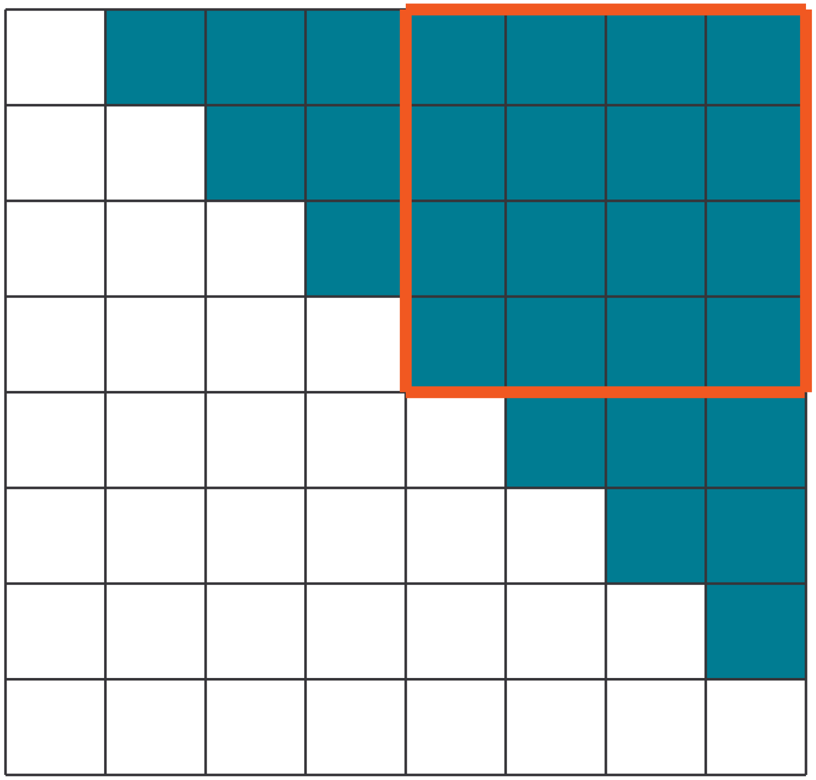
  upper left and lower right quadrants are finished.
- X* completes the dynamic program



Star Function Result

- A helper function
- Precondition:

  upper left and lower right quadrants are finished.
- $X*$ completes the dynamic program



$X*$

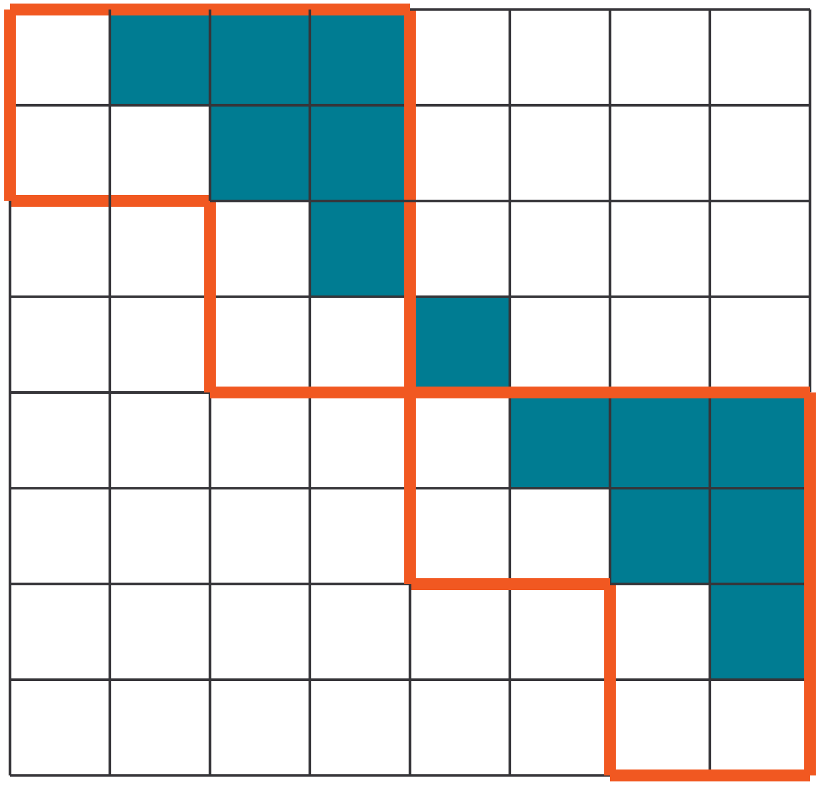
Valiant’s Algorithm

- First step:

  Two recursive calls

$$\begin{bmatrix} X_{11} & X_{12} \\ & X_{22} \end{bmatrix} := \begin{bmatrix} X_{11} & X_{12} \\ & X_{22} \end{bmatrix}^{+}$$

$$\begin{bmatrix} X_{33} & X_{34} \\ & X_{44} \end{bmatrix} := \begin{bmatrix} X_{33} & X_{34} \\ & X_{44} \end{bmatrix}^{+}$$

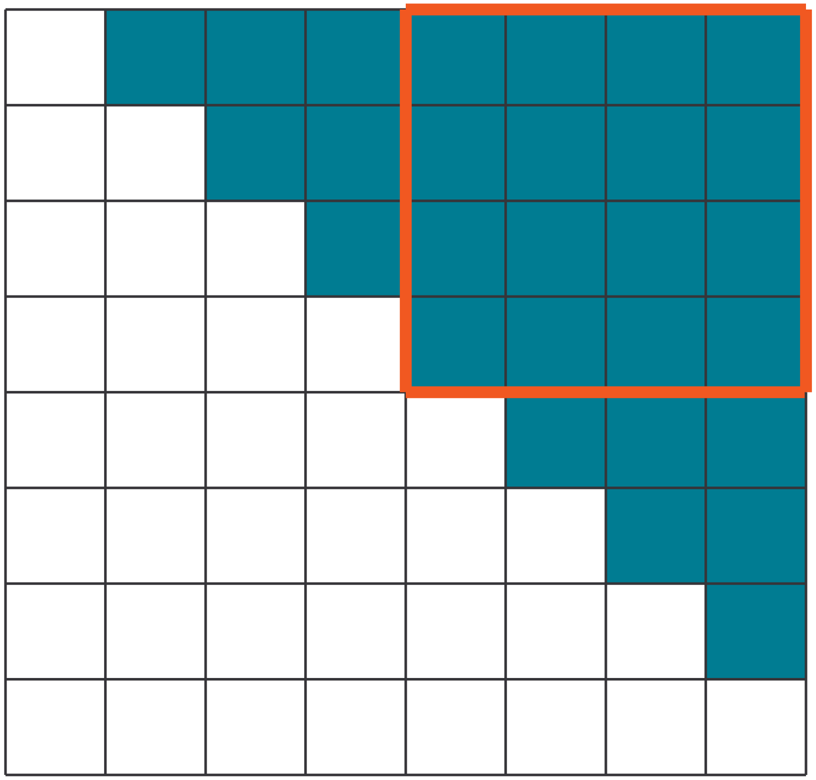


Valiant's Algorithm

- Second step: apply the Star Function

$$X := X^*$$

- The end of Valiant's algorithm
- But what is the Star Function?

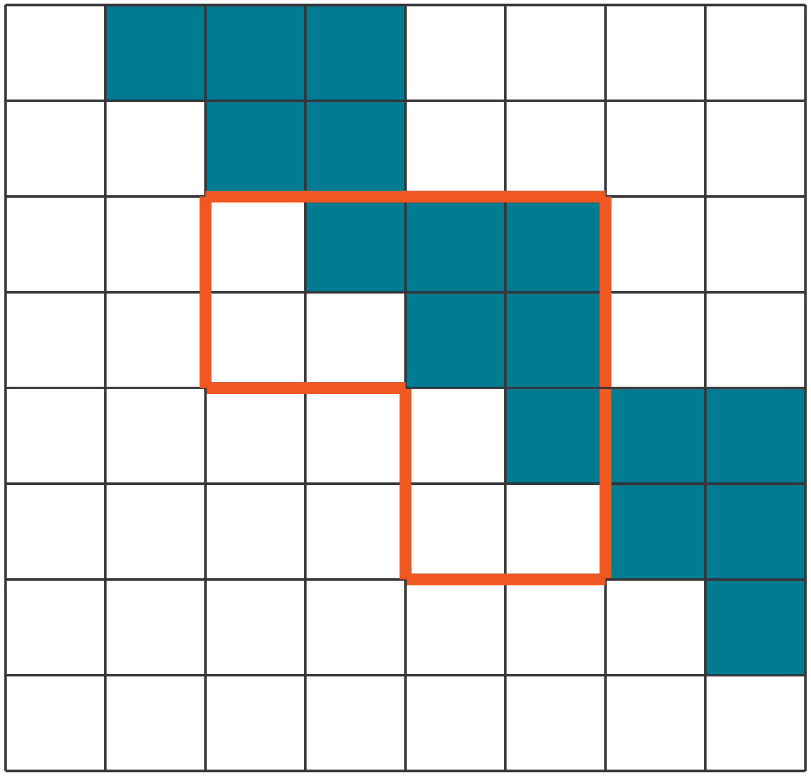


Star Function

- First step:

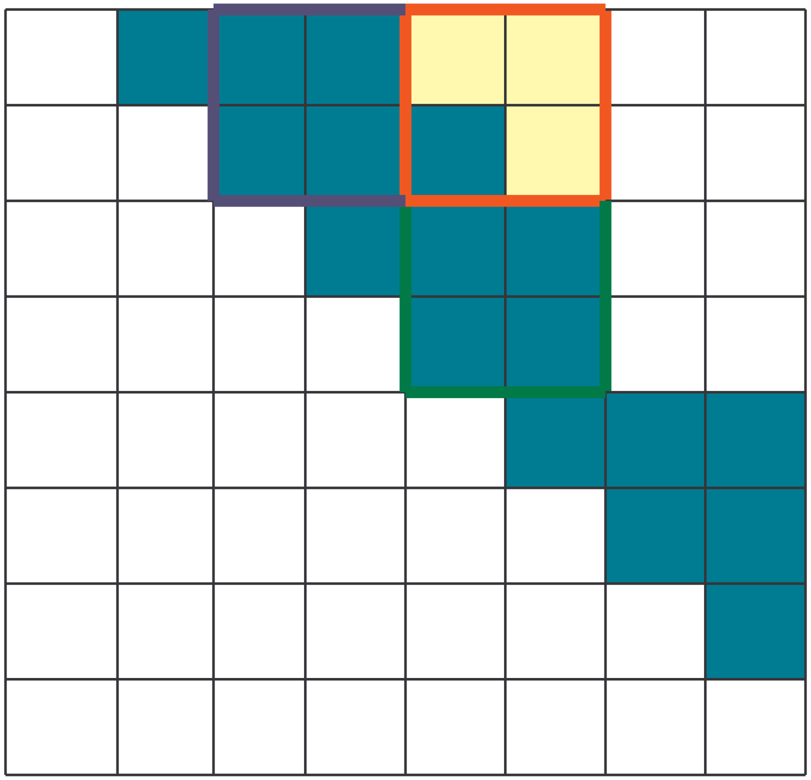
$$\begin{bmatrix} X_{22} & X_{23} \\ & X_{33} \end{bmatrix} := \begin{bmatrix} X_{22} & X_{23} \\ & X_{33} \end{bmatrix}^{*}$$

- Finishes the center quadrant



Star Function

- Second step:
- $X_{13} := X_{13} + X_{12} X_{23}$
- 🟨 means unfinished



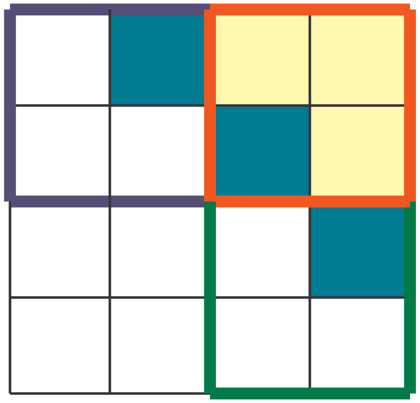
Star Function

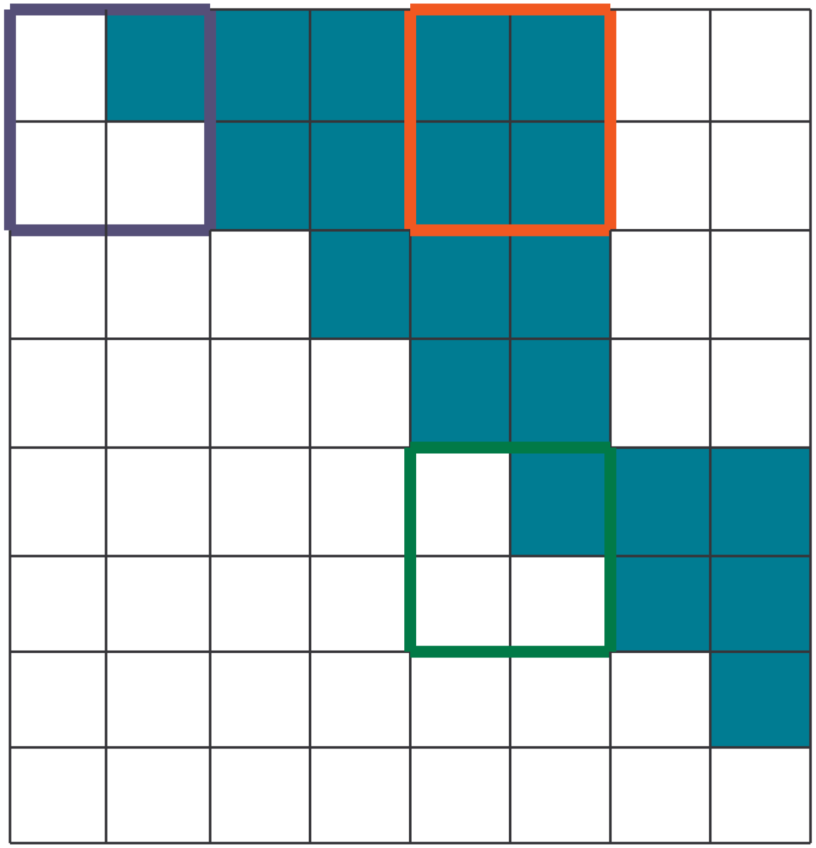
- Third Step:

  Recursive Star call

$$\begin{bmatrix} X_{11} & X_{13} \\ & X_{33} \end{bmatrix} := \begin{bmatrix} X_{11} & X_{13} \\ & X_{33} \end{bmatrix}^*$$

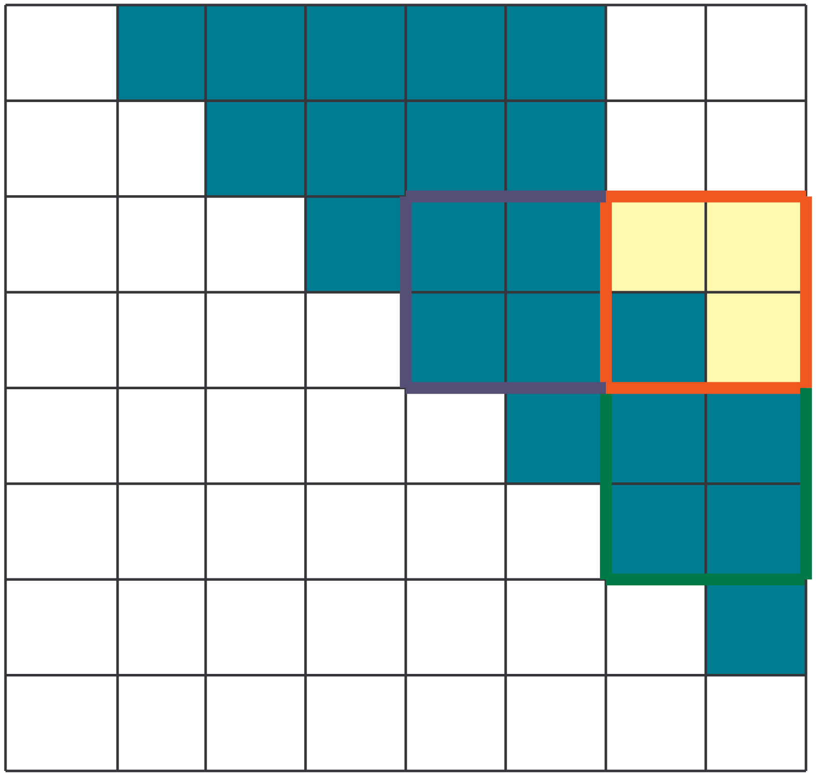
- Finishes X_{13}





Star Function

- Fourth step:
- $X_{24} := X_{24} + X_{23}\, X_{34}$
- 🟨 means unfinished



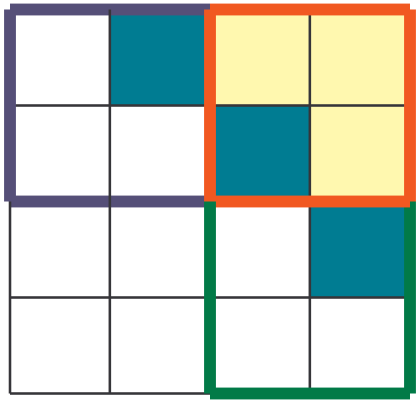
Star Function

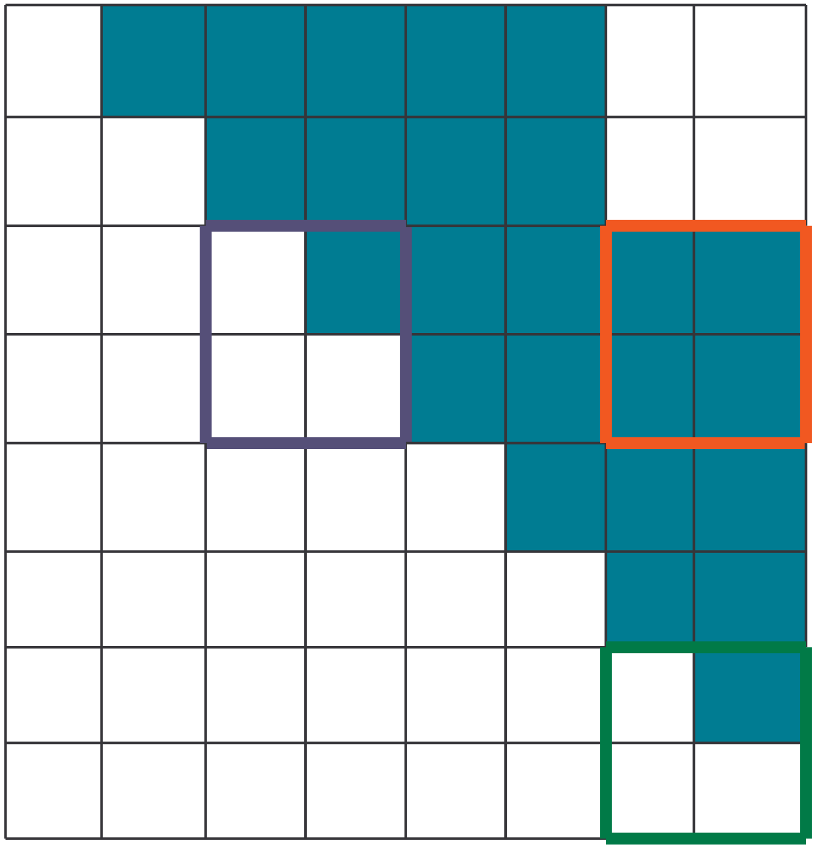
- Fifth Step:

  Recursive Star call

$$\begin{bmatrix} X_{22} & X_{24} \\ & X_{44} \end{bmatrix} := \begin{bmatrix} X_{22} & X_{24} \\ & X_{44} \end{bmatrix}^*$$

- Finishes X_{24}

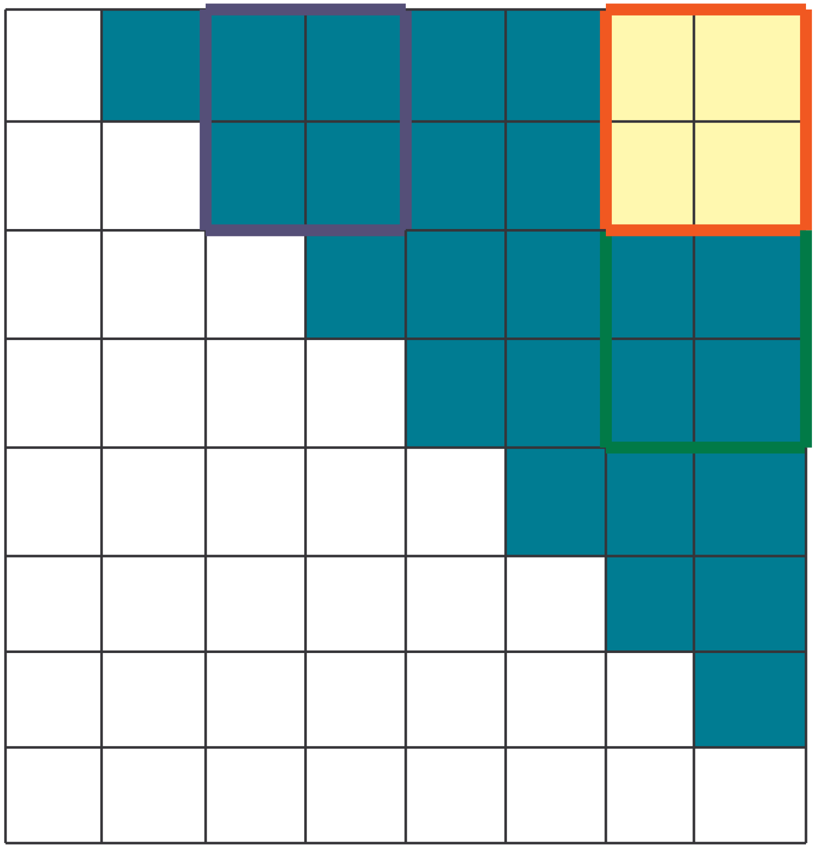




Star Function

- Fifth Step:

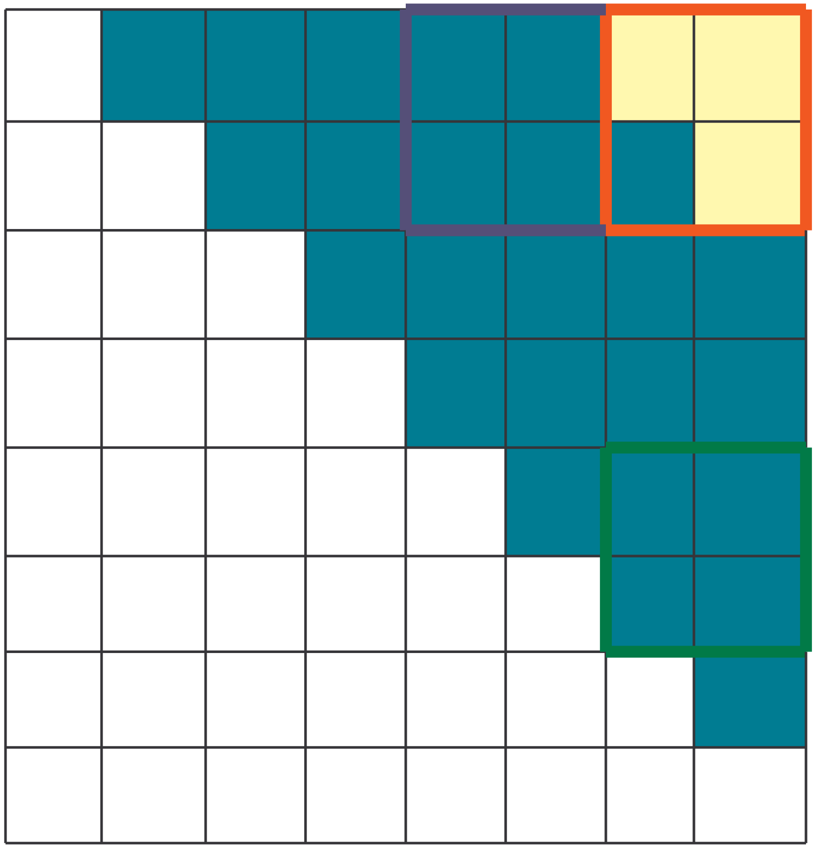
$X_{14} := X_{14} + X_{12}\, X_{24}$



Star Function

- Sixth Step:

$X_{14} := X_{14} + X_{13}\, X_{34}$



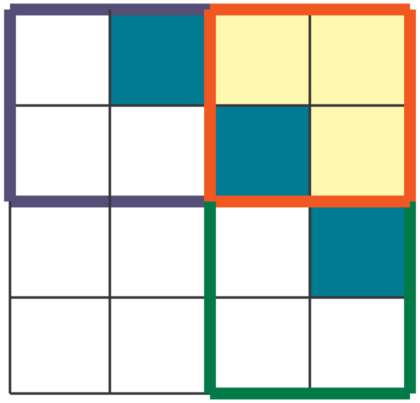
Star Function

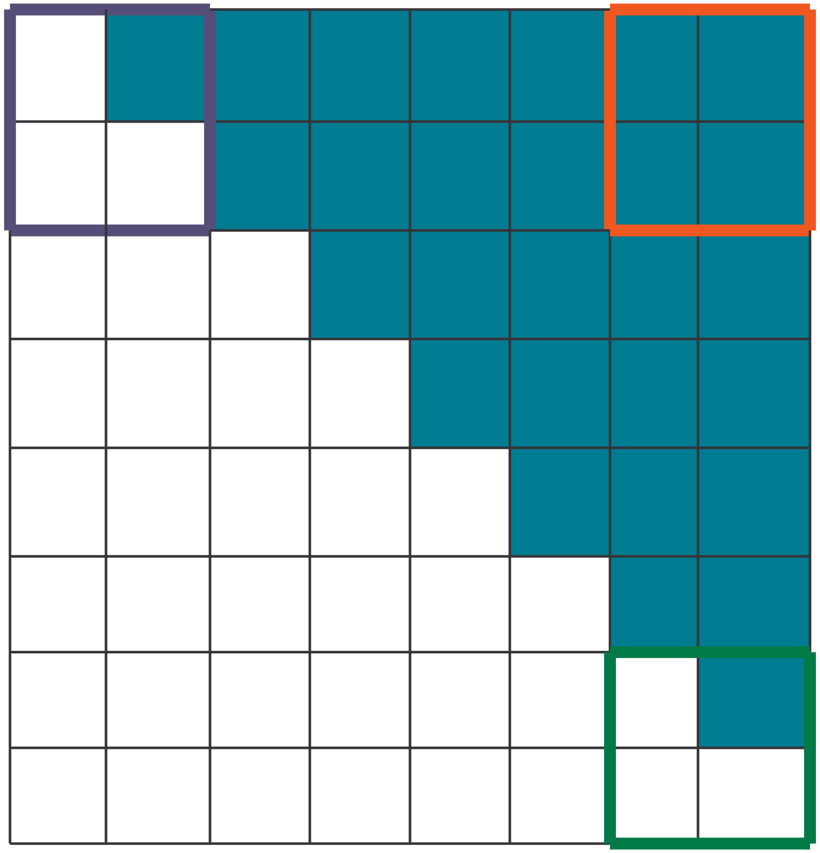
- Seventh Step:

  Recursive Star call

$$\begin{bmatrix} X_{11} & X_{14} \\ & X_{44} \end{bmatrix} := \begin{bmatrix} X_{11} & X_{14} \\ & X_{44} \end{bmatrix}^{*}$$

- Finishes X_{44}





Time bounds

- n refers to matrix size not problem size
- $T(n)$: Valiant’s algorithm
- $S(n)$: Star function
- $M(n)$: Block Matrix Multiplication
- All are $O(n^3)$

$$T(n) \le 2T(n/2) + S(n)$$
$$S(n) \le 4S(n/2) + 4M(n/4)$$
$$M(n) \le 8M(n/2)$$

Cache-aware Algorithms

- Cache-oblivious algorithms do not depend on cache parameters
- Cache-aware algorithms have a tuning parameter depending on cache size, line size, etc

Blocked Valiant’s Algorithm

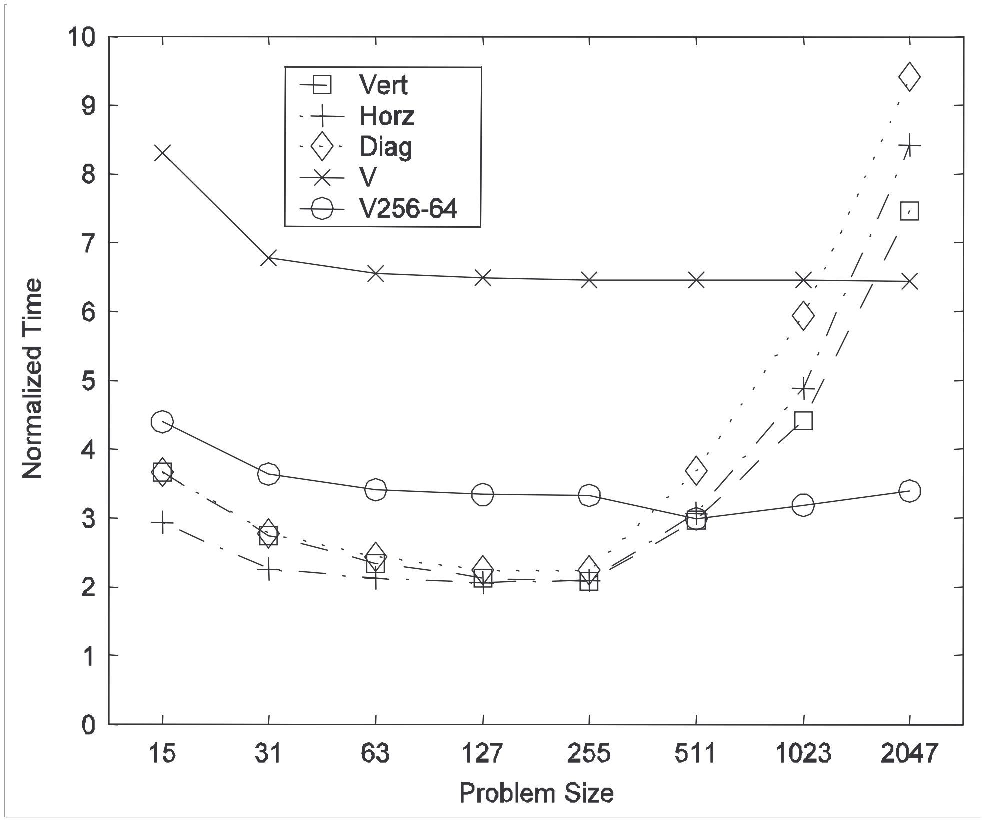
- Valiant’s algorithm incurs overhead from recursive calls and blocked matrix multiplication
- Recursion unnecessary for small problems
- Make Valiant’s algorithm cache-aware
- If matrix is sufficiently small use
  - normal matrix multiplication
  - CKY

Experiments

- Compared
  - Valiant’s algorithm
  - Blocked Valiant’s algorithm
  - Three variants of standard dynamic programming
- Time comparison
- Instruction Count
- Cache simulations using Valgrind
- 1 GHz AMD Athlon
  - 64 KByte L1 data cache (2-way)
  - 256 KByte L2 data cache (16-way)

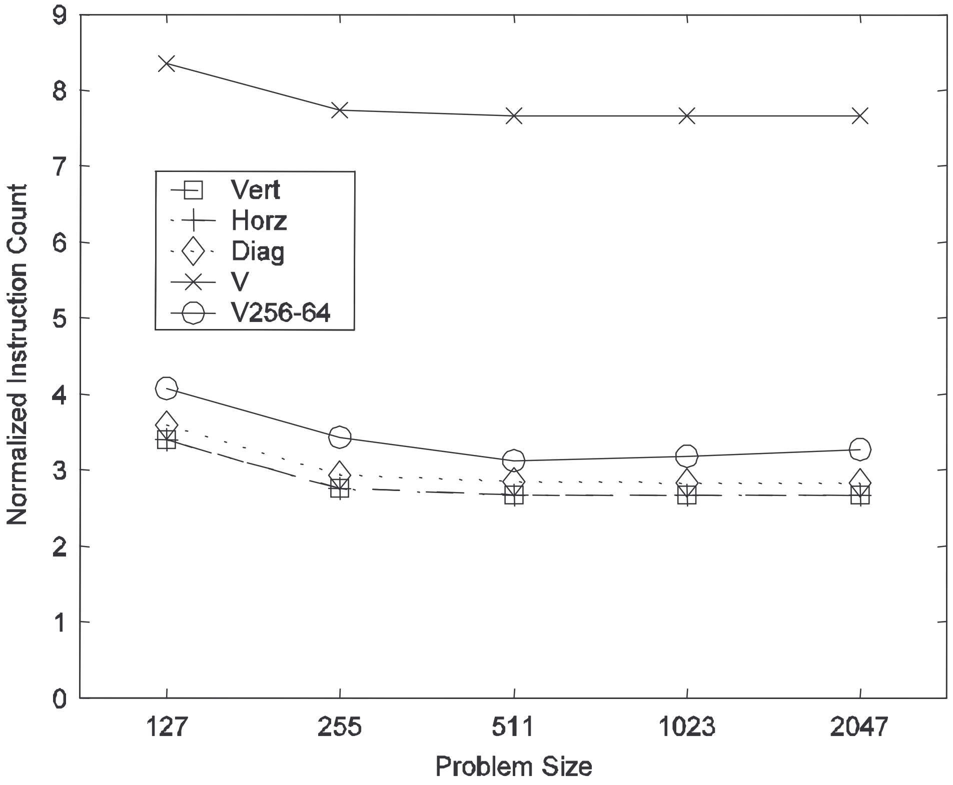
Time Comparison

- Normalized Time = Time/n^3
- V256-64
  - Blocked Valiant's algorithm
  - $n \le 256$ use CKY
  - $n \le 64$ use standard matrix multiplication



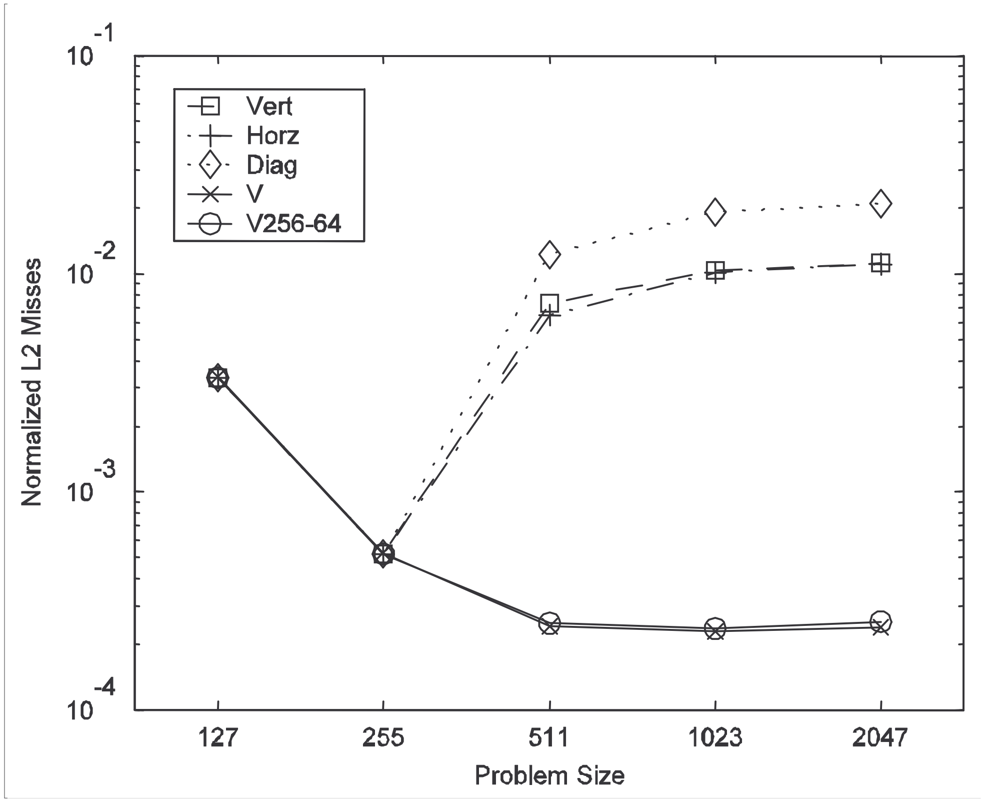
Instruction Count

- Valiant's algorithm uses the most instructions
- Blocked Valiant's algorithm is near CKY



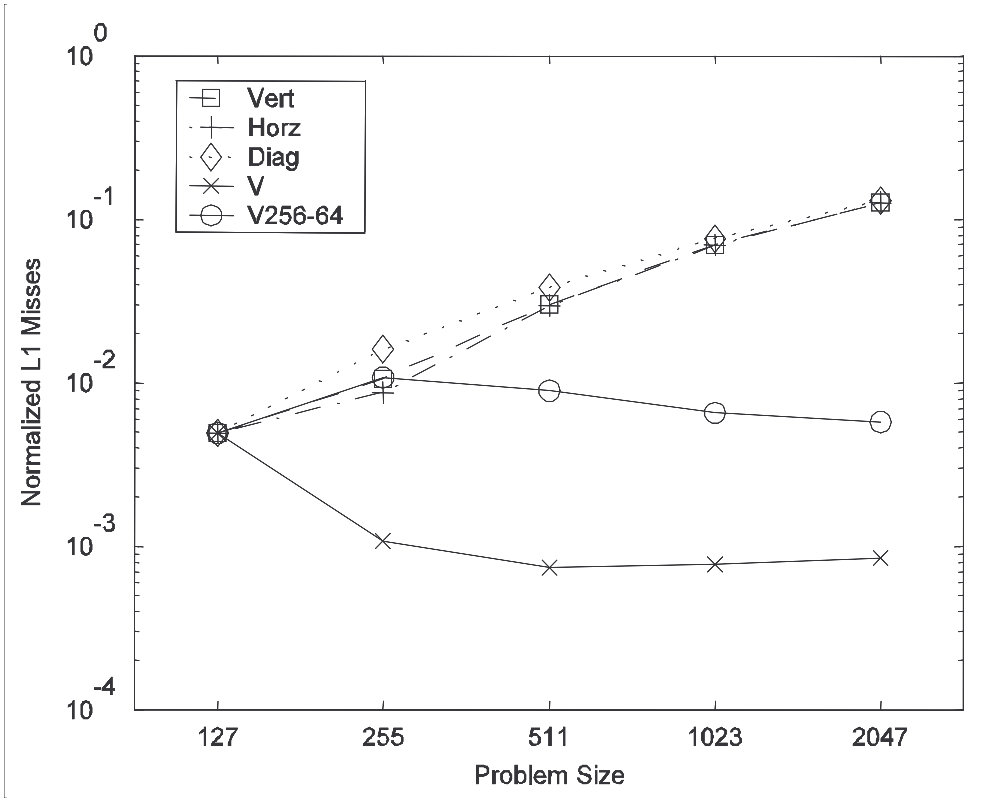
L2 Cache Misses

- Nearly 100 times more L2 cache misses in standard dynamic programming



L1 Cache Misses

- Blocked Valiant's algorithm trades instructions for more L1 cache misses



Conclusion

- Instruction count not the only important thing
- Cache misses matter
- Divide and conquer gives good cache behavior

Automatic Tactilization of Graphical Images

With

Matt Renzelman

Satria Krisnandi

The Tactilization Problem

- Graphical images are heavily used in math, science and engineering textbooks and papers
  - Line graphs and bar charts
  - Diagrams
  - Illustrations
- Tactual perception is the best modality for the blind to understand such images
- Tactilization of graphical images
  - Currently done manually
  - Labor-intensive and time consuming
  - How much of this process can be automated?

Outline

- Tactual Perception
- Overview of tactilization process
- Text segmentation
- Braille text placement
- Other subprojects
- Demonstration

Tactile Perception

- Resolution of human fingertip: 25 dpi
- Tactual field of perception is no bigger than the size of the fingertips of two hands
- Color information is replaced by texture information
- Visual bandwidth is 10^6 bits per second, tactile is 10^2 bits per second

Braille

- System to read text by feeling raised dots on paper (or on electronic displays). Invented in 1820s by Louis Braille, a French blind man.

a	⠁	b	⠃	c	⠉	z	⠵
and	⠯	the	⠮	with	⠾	mother	⠐⠍
th	⠹	ch	⠡	gh	⠣		
Z	⠠⠵	3	⠼⠉				

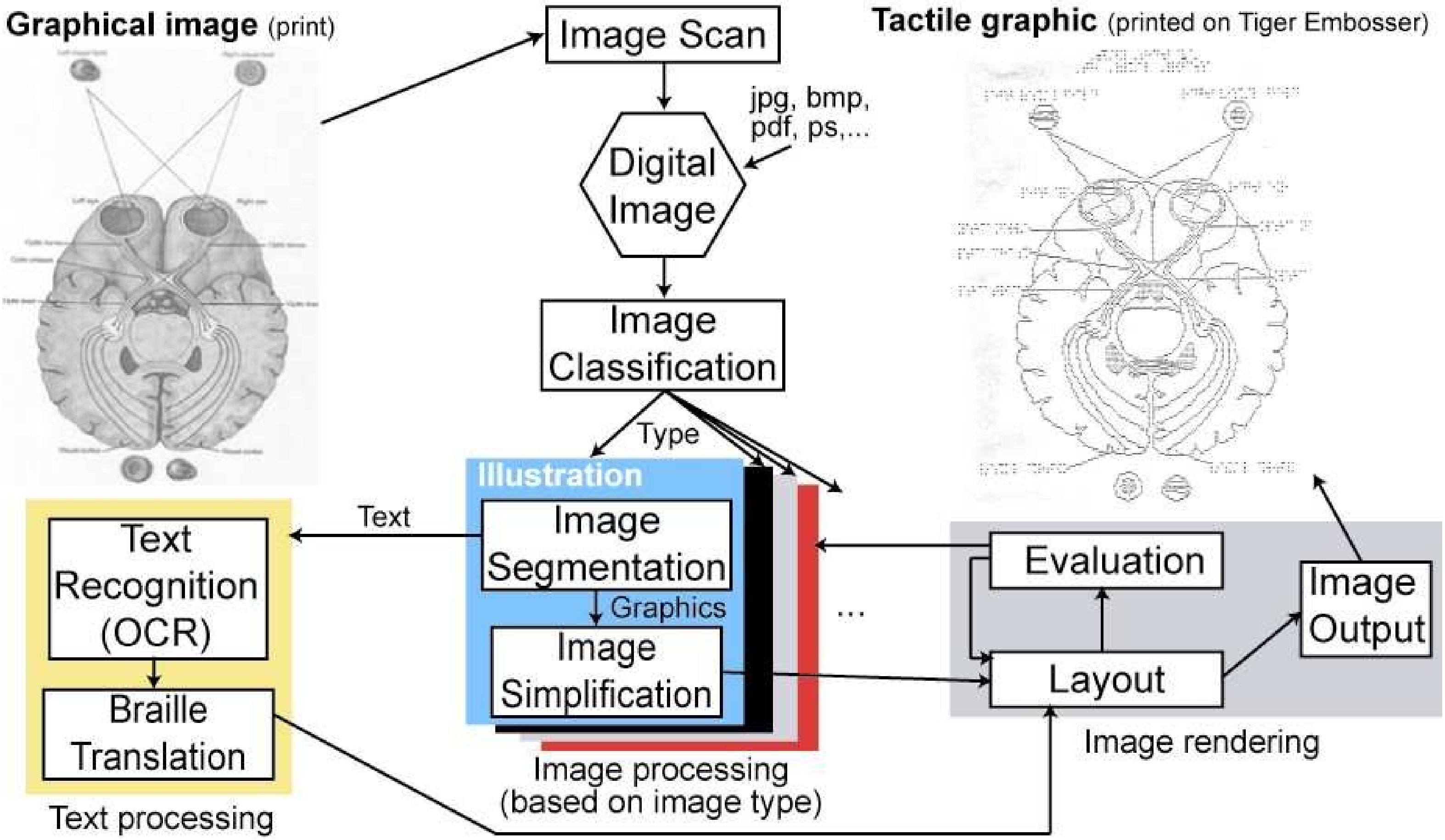
Critical fact: Fixed height and width

Mode characters: cap and num.

Tiger Embosser

- 20 dpi (raised dots per inch)
- 7 height levels (only 3 or 4 are distinguishable)
- Prints Braille text and graphics
- Prints dot patterns for texture
- Invented by a blind man, John Gardner

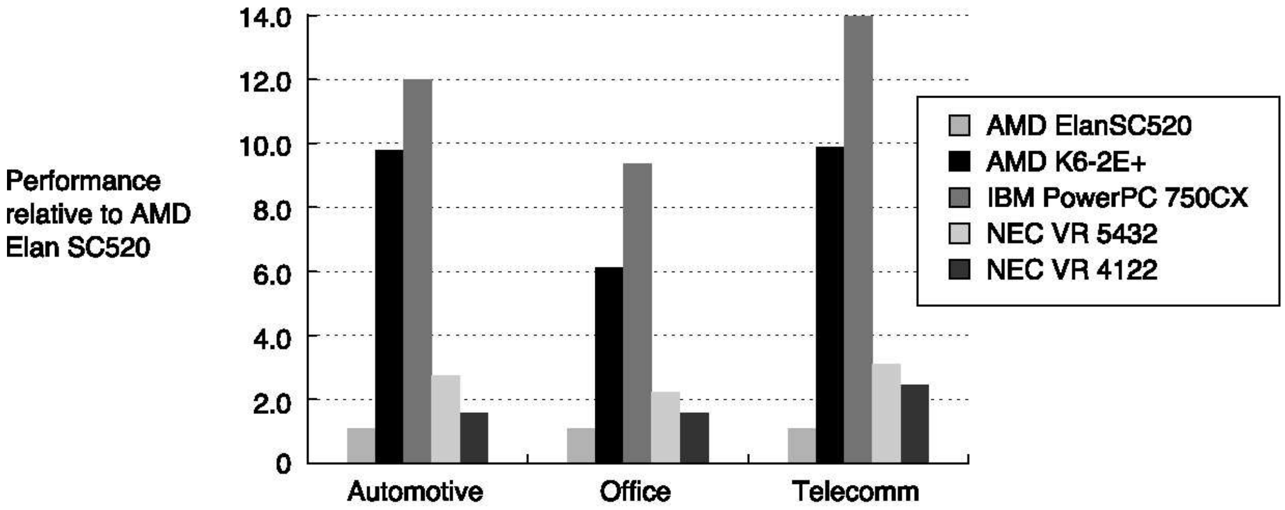
Automatic Tactilization Process



Key Problems

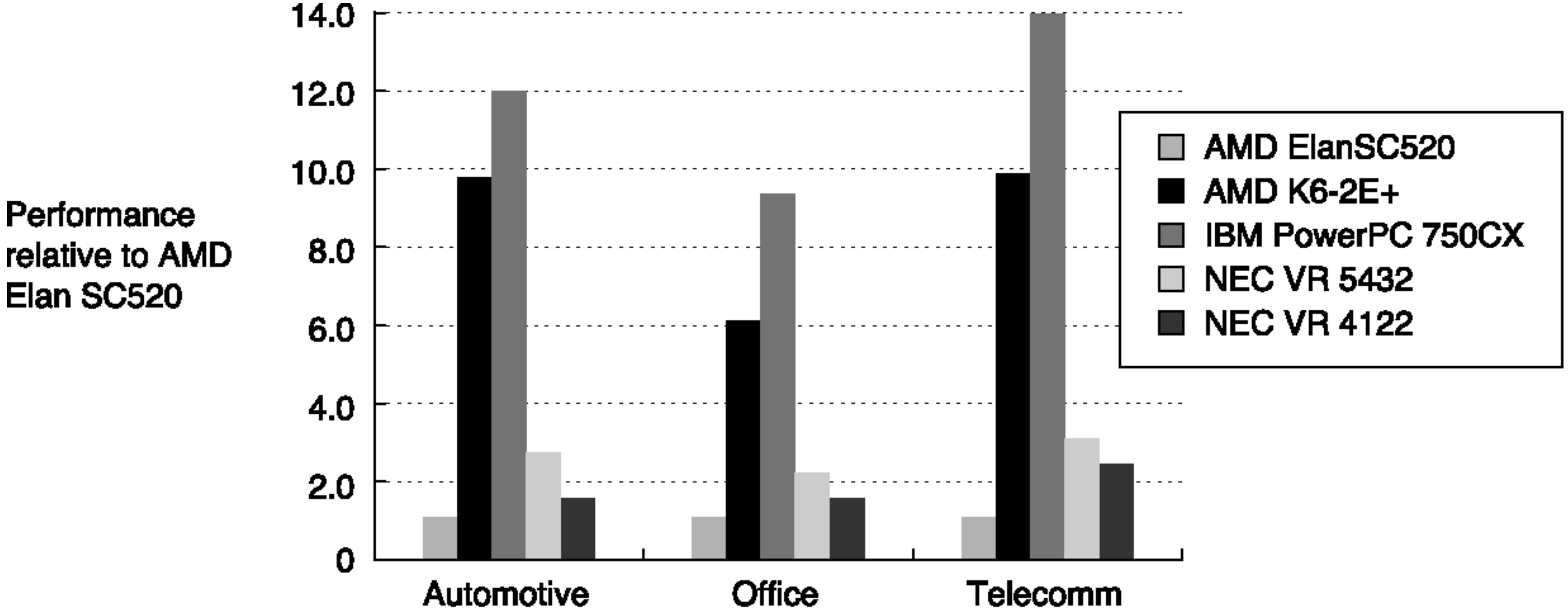
- Graphical images meant for the visual mode must be modified for the tactual mode.
  - Text ⇒ Braille
  - Colors ⇒ replace with textures or reduce number
  - Area ⇒ Larger for Braille text to fit
  - Resolution ⇒ Lower to 20 dpi
  - Shading or 3-D effects ⇒ replace with outlines
  - Noise ⇒ remove noise, enhance contrast
- Classification of graphical images for mass production
  - Images in the same class require similar processes.

Example



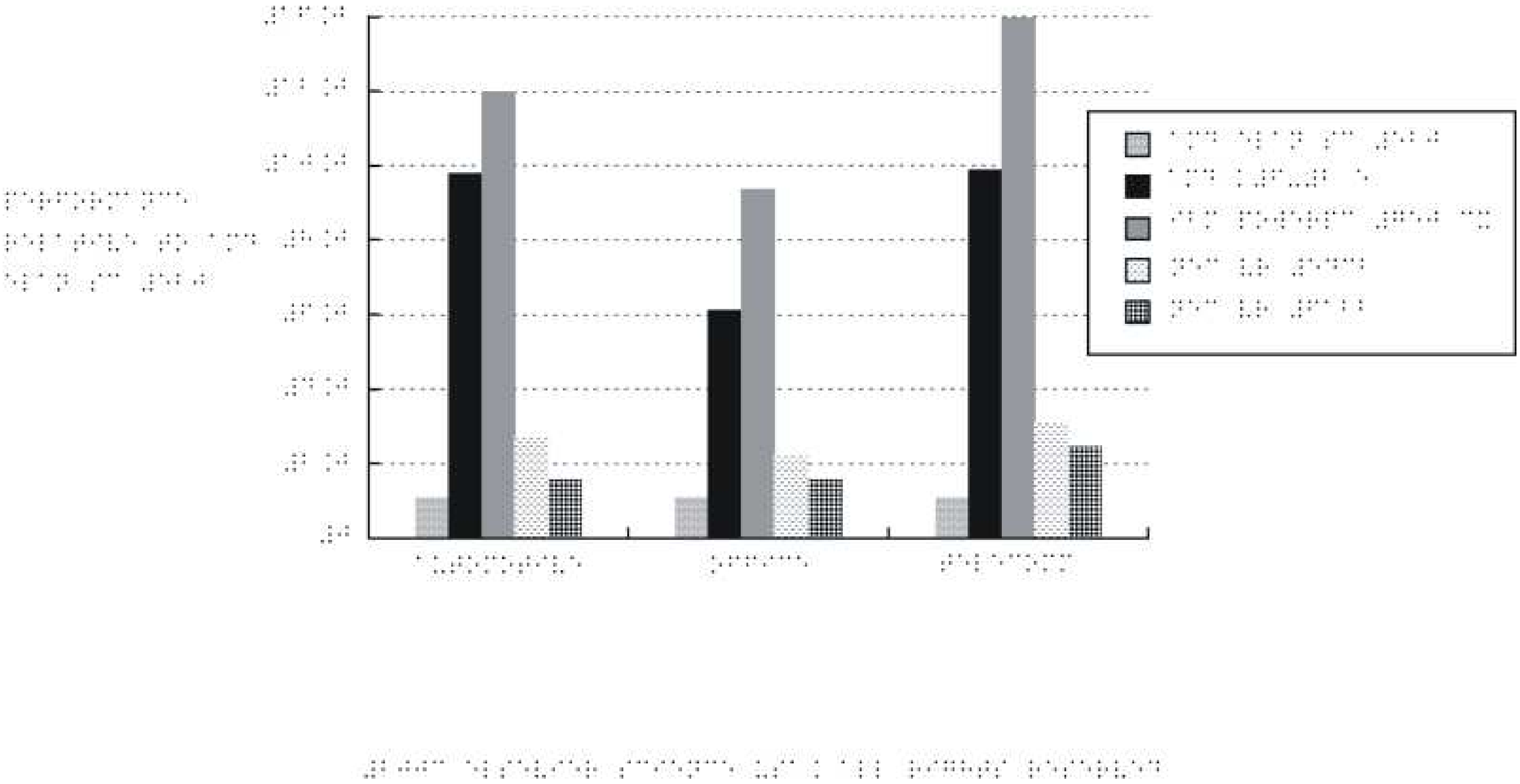
From *Computer Architecture, A Quantitative Approach, Third Edition*, by Hennessy and Patterson.

Sample image

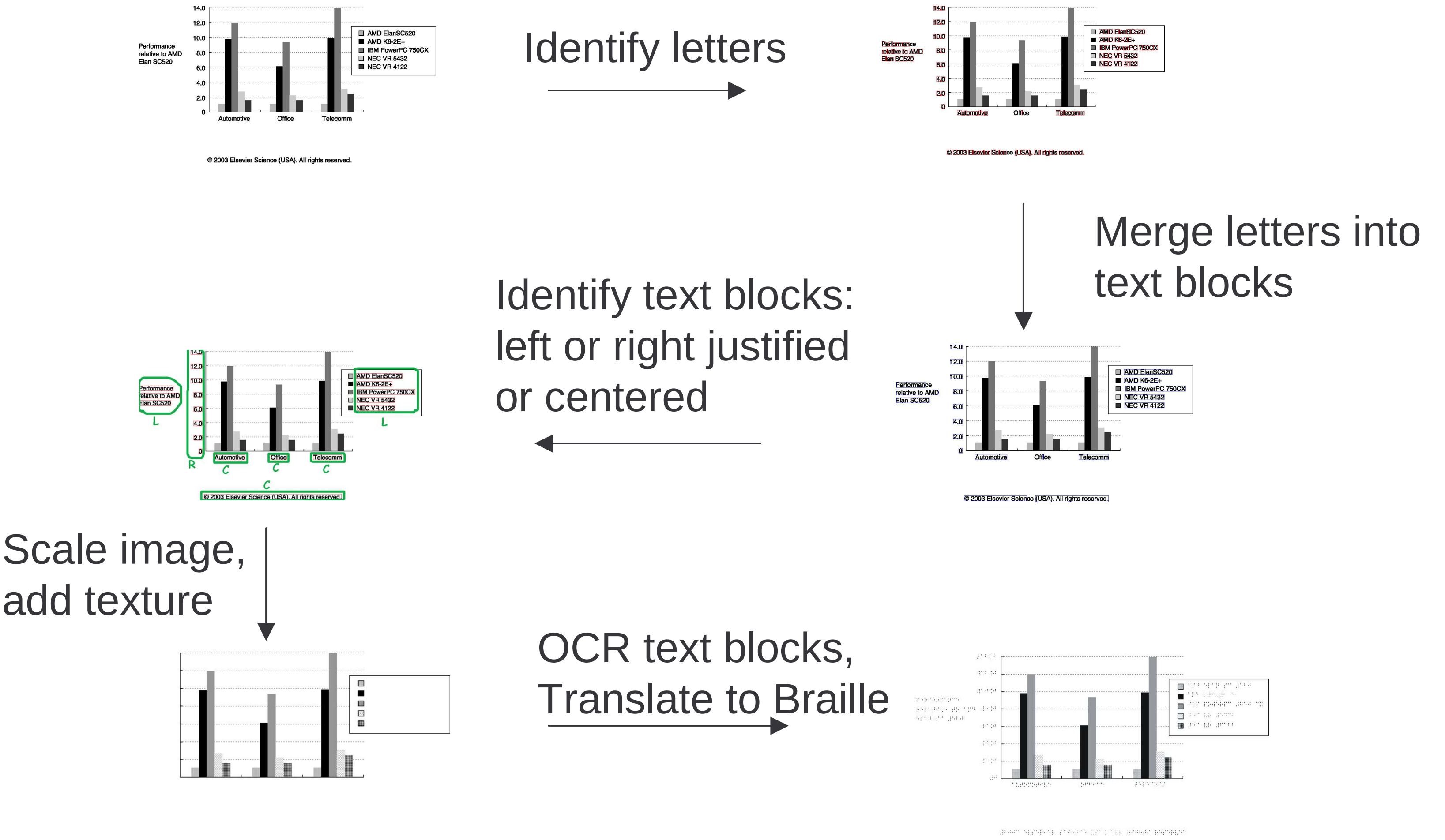


From *Computer Architecture, A Quantitative Approach, Third Edition*, by Hennessy and Patterson.

Desired result



Overall Process



Identify letters

Merge letters into text blocks

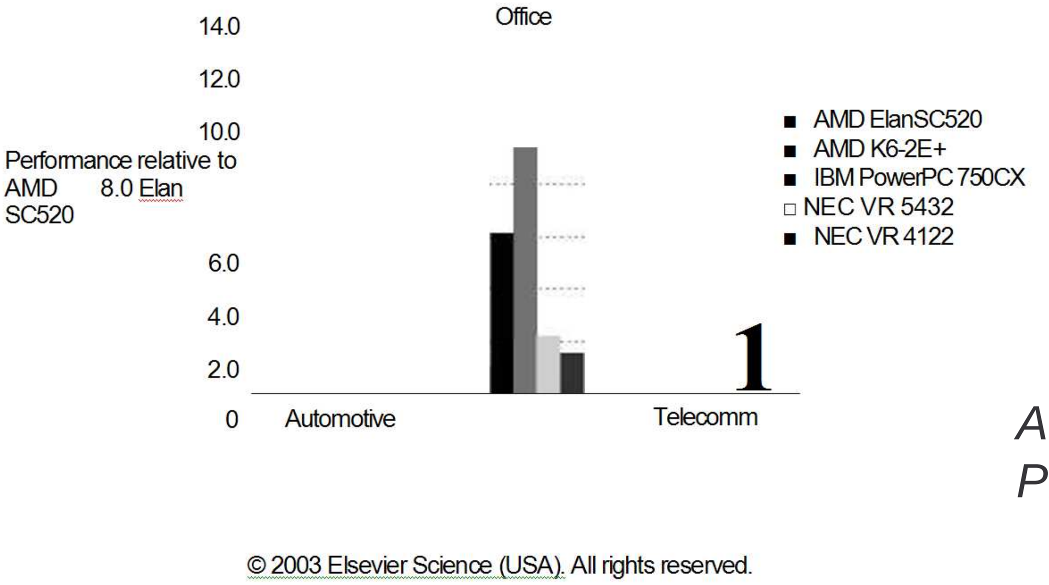
Identify text blocks: left or right justified or centered

Scale image, add texture

OCR text blocks, Translate to Braille

Finding Text

- Why not just use standard optical character recognition (OCR)?
  - OCR is not effective for graphical images.



ABBYY FineReader 7.0 Professional Edition

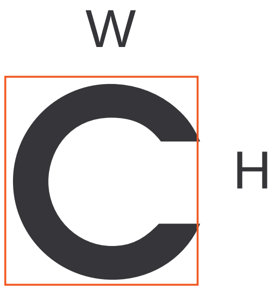
Finding Text Letters

- Uses the following principles
  - Text in an image is usually in one font
  - Fonts are designed to have a uniform density at a distance.
  - In the absence of noise an individual letter tends to be connected component of one color. Exceptions are i and j.
- Train on some simple features of letters. Connected components with similar features are also letters.

Features

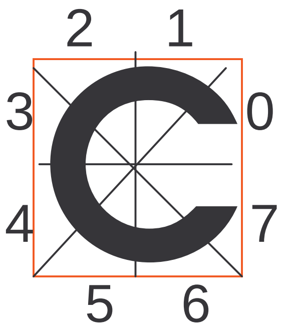
Century Gothic

W = width of bounding box
H = height of bounding box
A = area of bounding box
R_i = i-th radial slice density



A = W • H

R_i = number of black pixels in i-th slice where a slice is an angle of 360/n. The total number of slices is n.



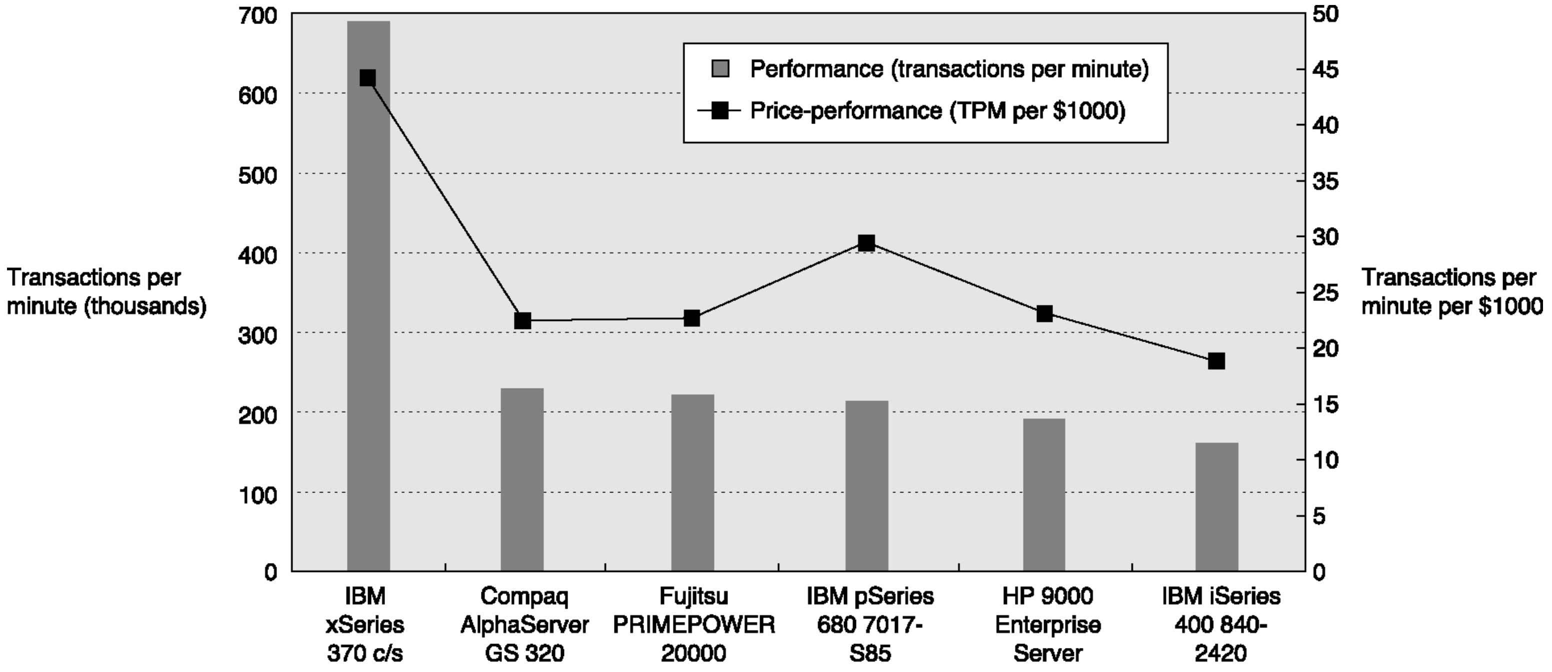
Center is center of mass of black pixels

Training/Finding

- Training:
  - Sample the connected components and compute their features.
- Finding:
  - For a new connected component compute its features.
  - If there is a close enough match of features with some member of the database then declare the component to be a letter.
- Parameters
  - How close is close enough
  - How many slices

Step 1: training

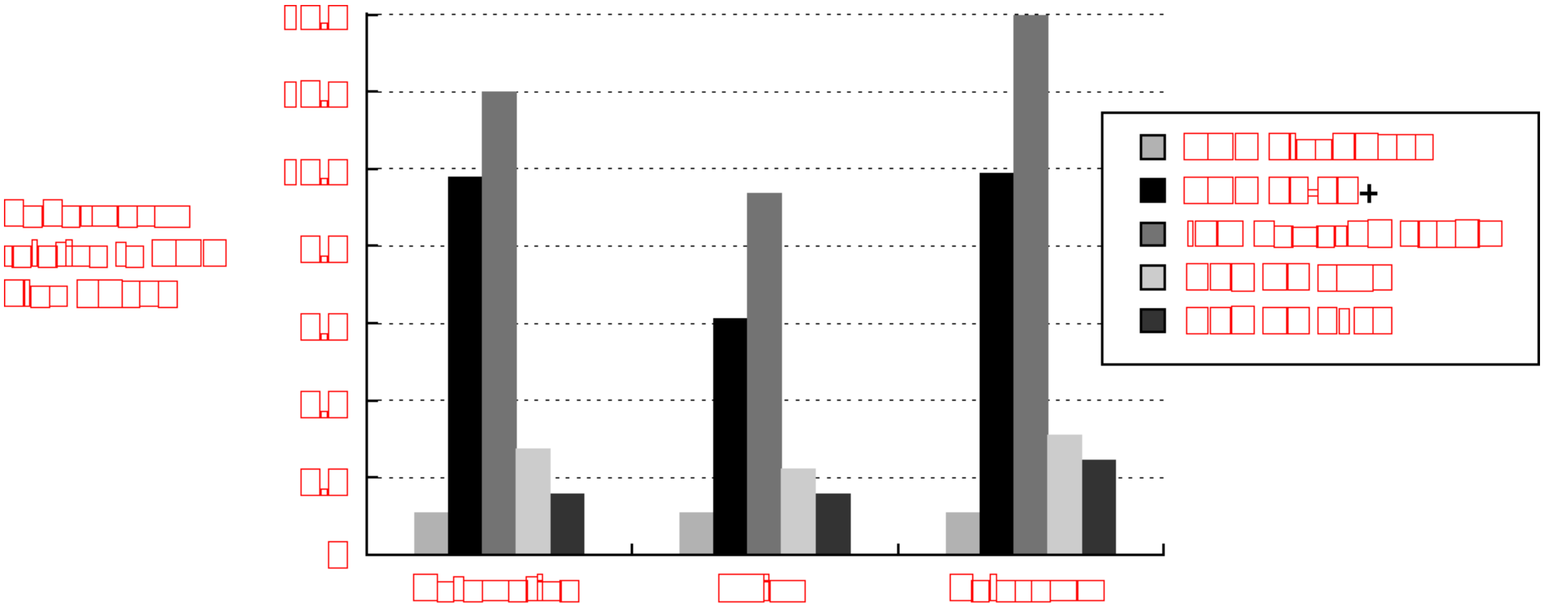
- Number of components in training set: 271



Ch.1, Figure 23

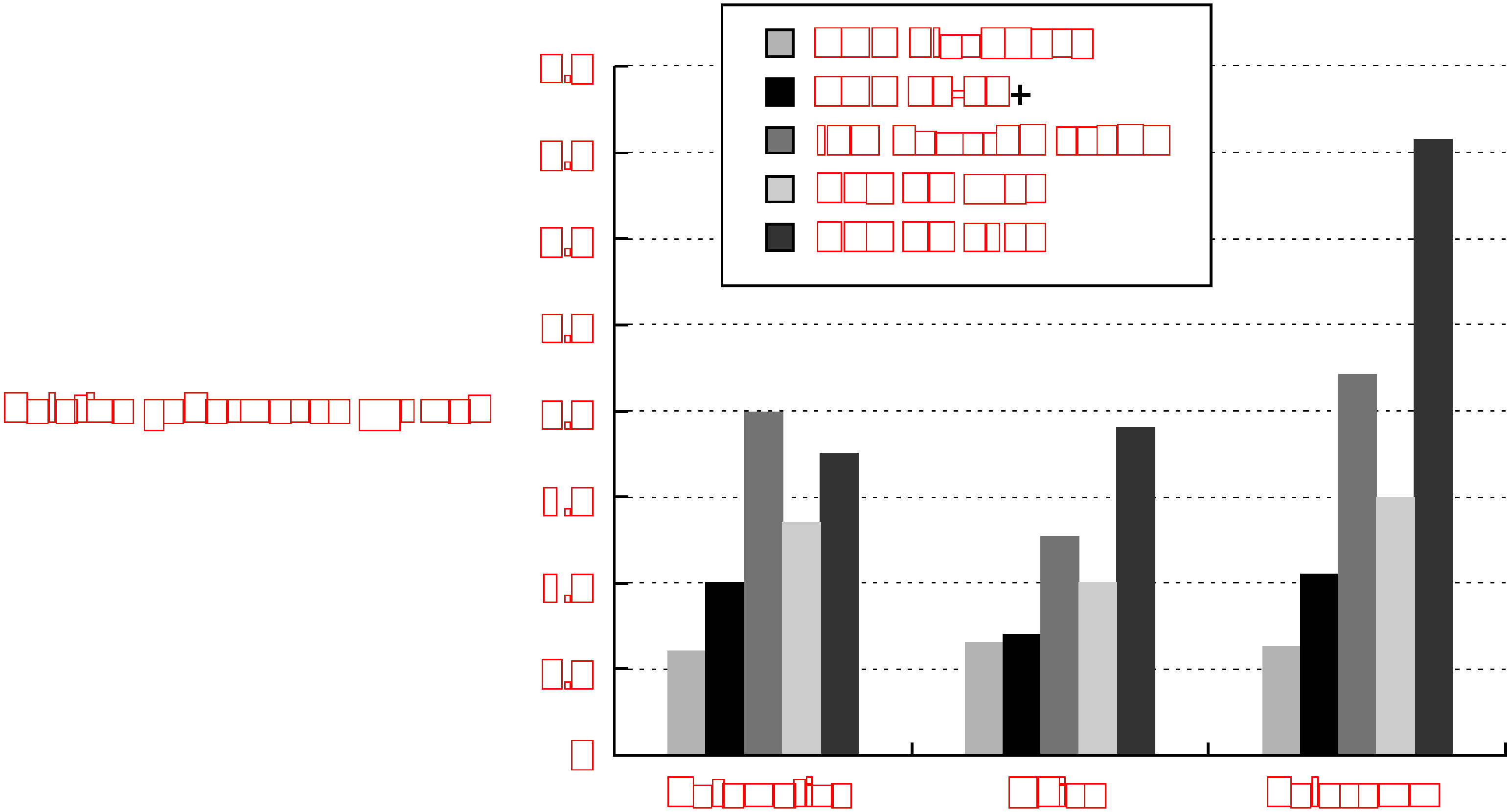
Step 2: results (1/3)

- No false positives



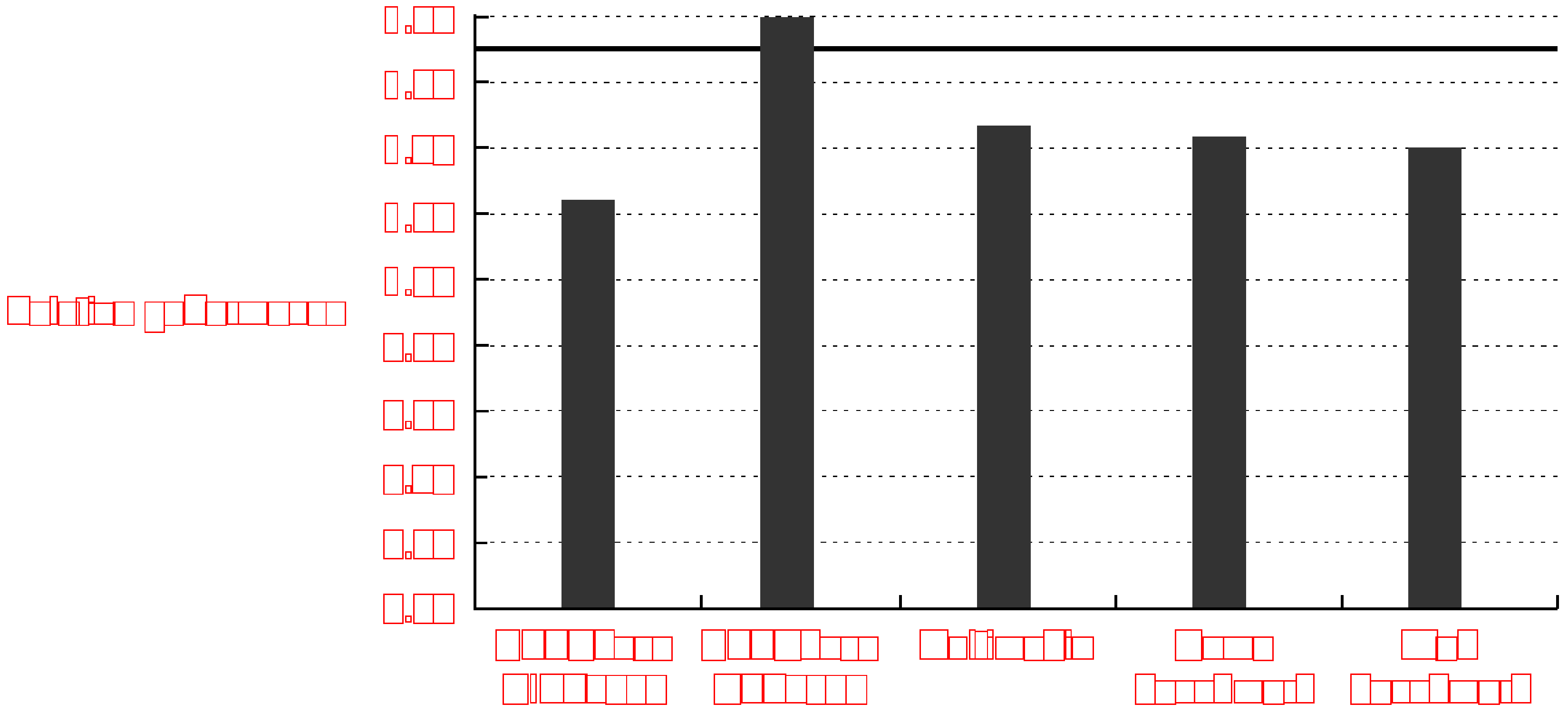
Ch.1, Figure 25

Step 2: results (2/3)



Ch.1, Figure 27

Step 2: results (3/3)

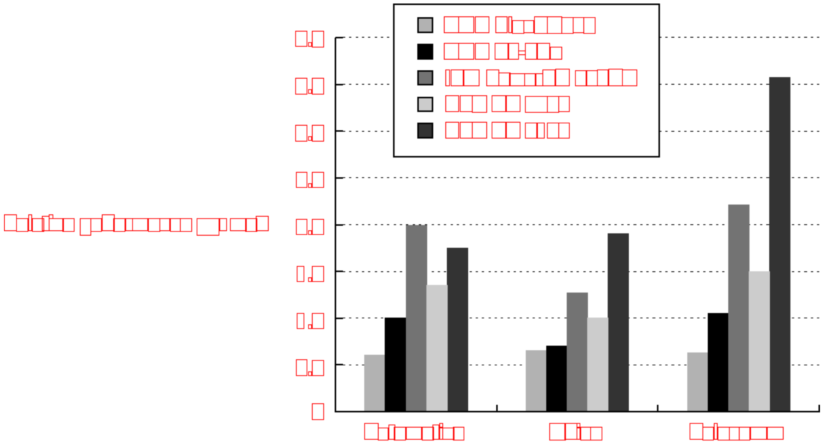


Ch.1, Figure 28

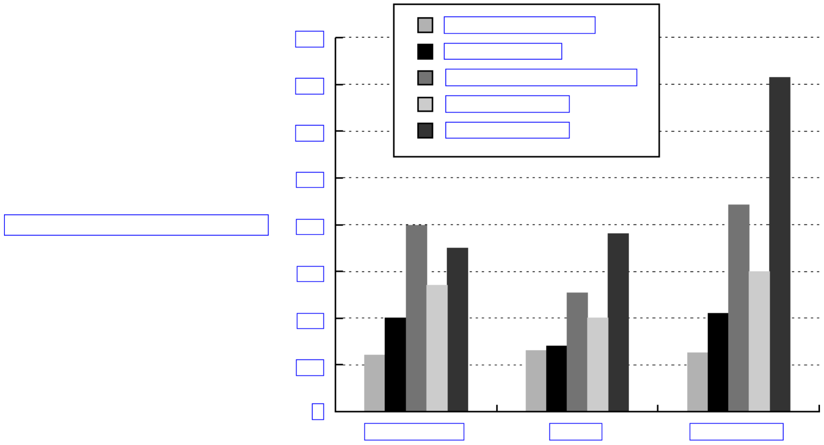
Finding Text Blocks

- Principles
  - Most text tends to be in horizonal lines
  - Some text is vertical
  - Some text is diagonal
- We are developing methods that find lines using the centroids of the letters found.
  - Minimum spanning tree
  - Merge test using linear regression

Group characters logically



[illegible]



Group characters logically

- Extracting a set of isolated characters from an image is insufficient
  - Need groups of Braille characters for easier placement
- Challenges
  - Text can be at many angles
  - Individual characters may be aligned along multiple axes

Our approach

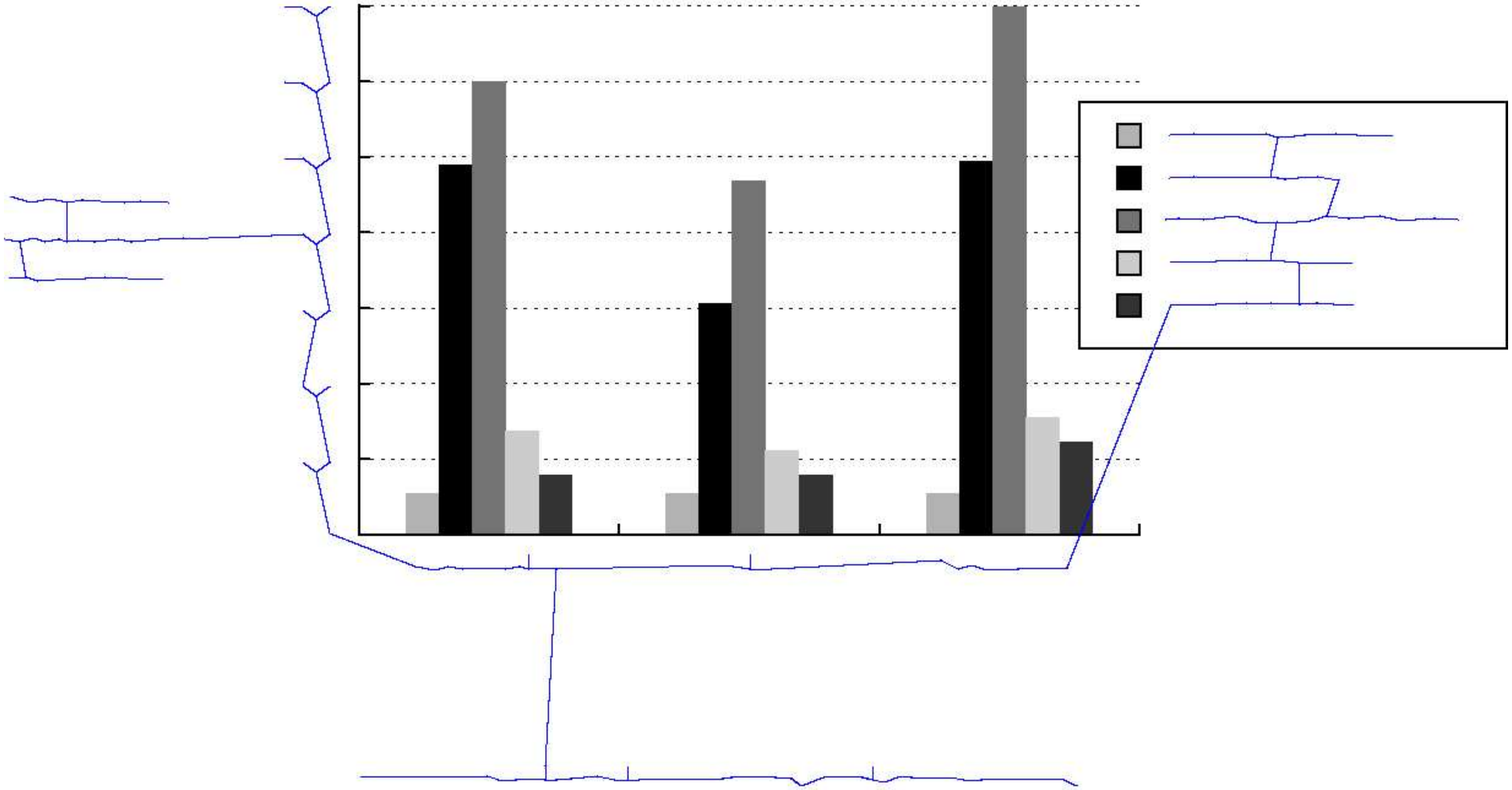
- Step 1: User provides training set
  - Software examines defining characteristics
- Step 2: Automatically find similar groups in remaining images
  1. A. Minimum spanning tree
  2. B. Discard useless edges
  3. C. Discard inconsistent edges
  4. D. Create merged groups

Defining characteristics

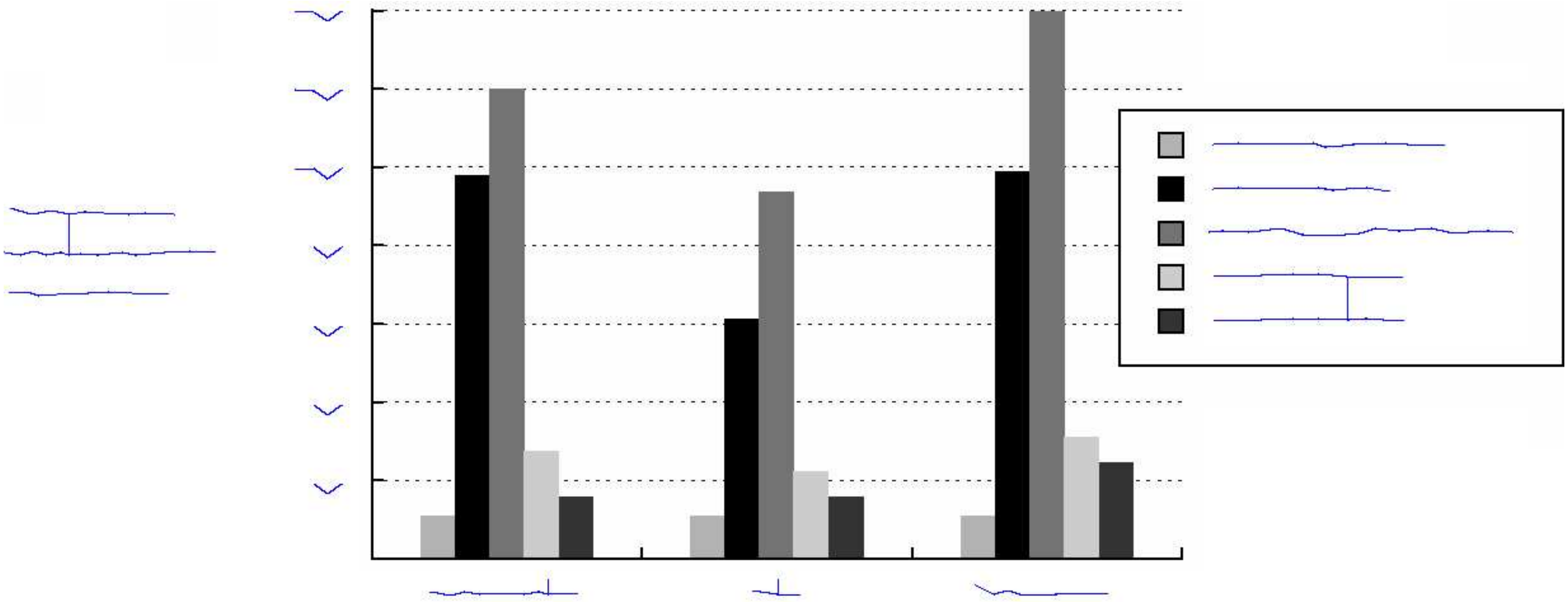
- Inter-character spacing
- Line of best fit
    - Perpendicular regression vs. linear regression
    - Mean squared error
    - Angle

Minimum spanning tree (1)

- Treat the centroid of each connected component as a node

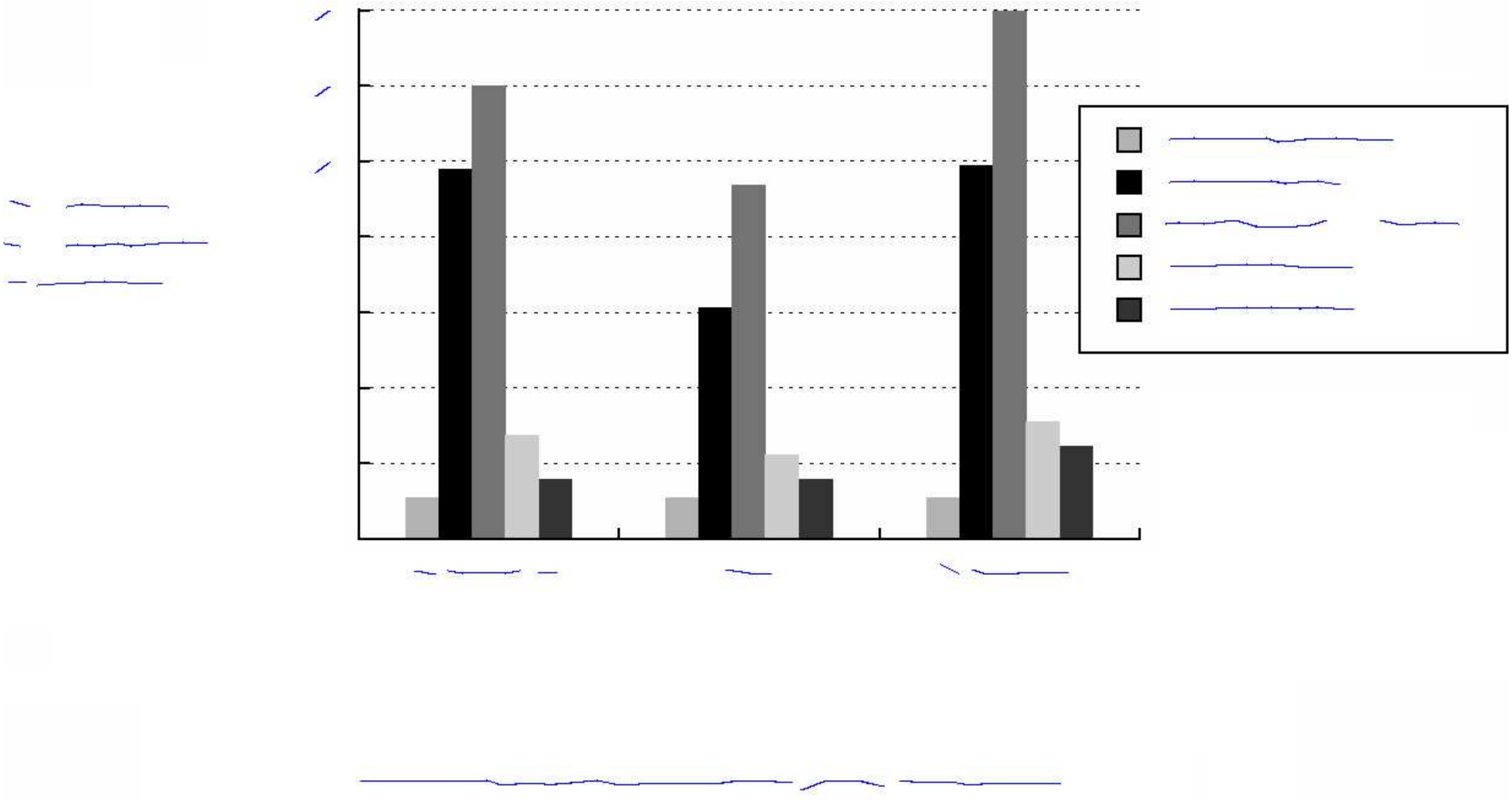


Discard useless edges (2)



[illegible]

Discard inconsistent edges (3)



Final merge step (4)

Merge only if the resultant group is consistent

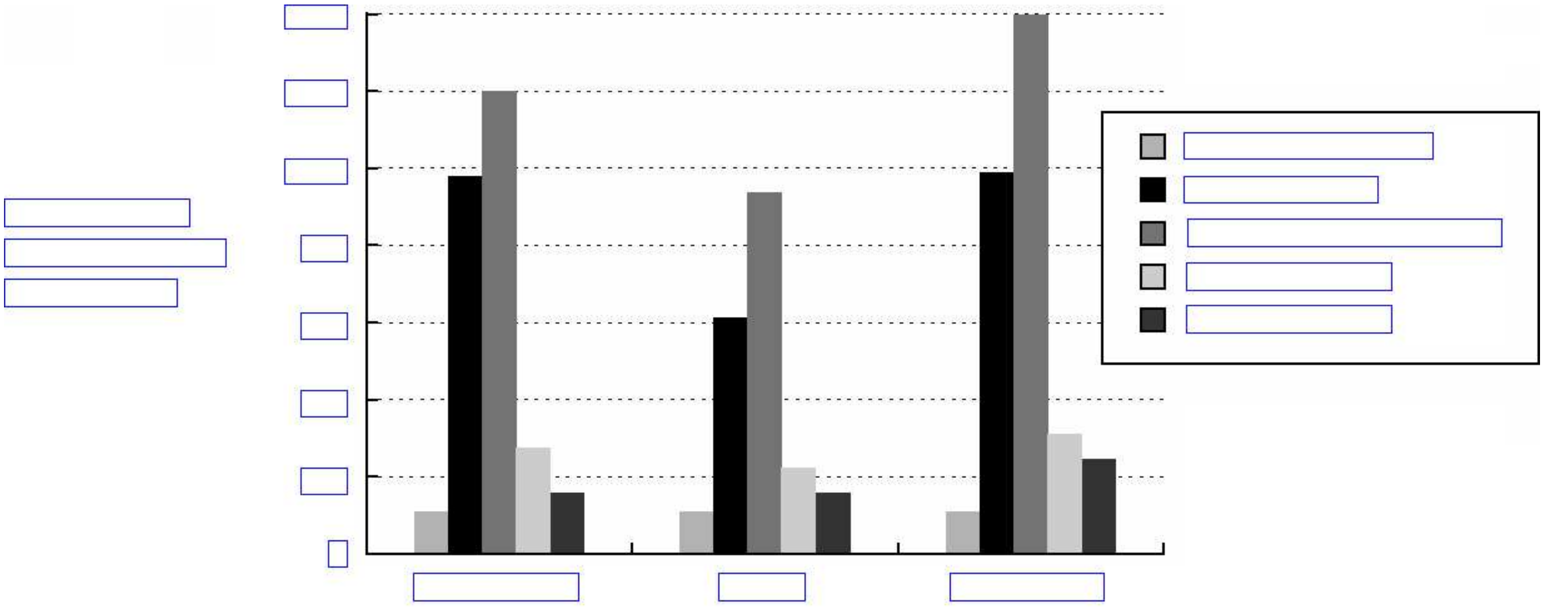
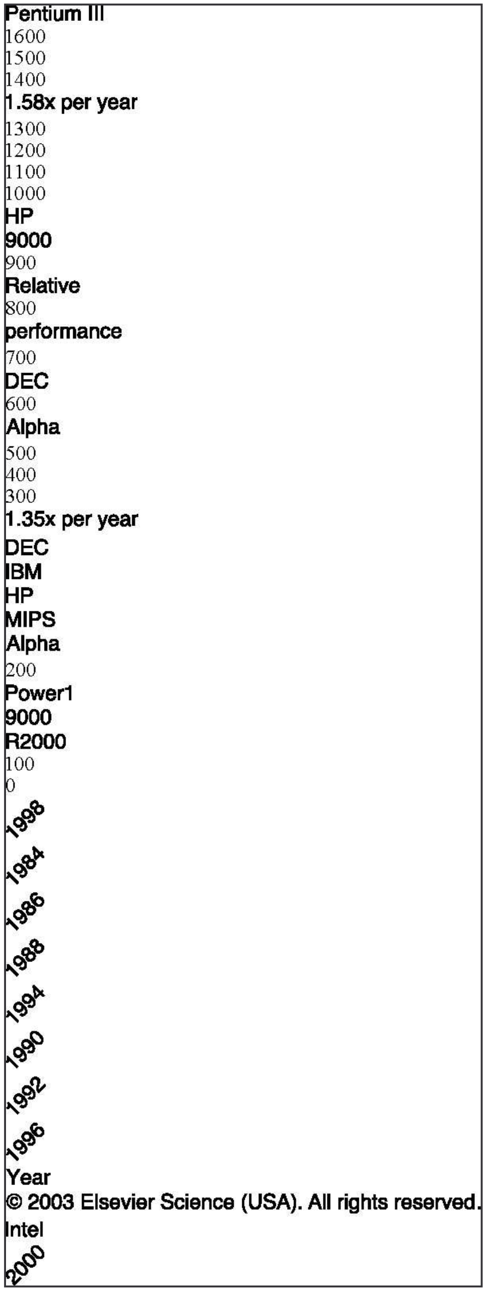


Image of
text boxes



Pentium III
1600
1500
1400
1.58x per year
1300
1200
1100
1000
HP
9000
900
Relative
800
performance
700
DEC
600
Alpha
500
400
300
1.35x per year
DEC
IBM
HP
MIPS
Alpha
200
Power1
9000
R2000
100
0
1998
1984
1986
1988
1994
1990
1992
1996
Year
© 2003 Elsevier Science (USA). All rights reserved.
Intel
2000

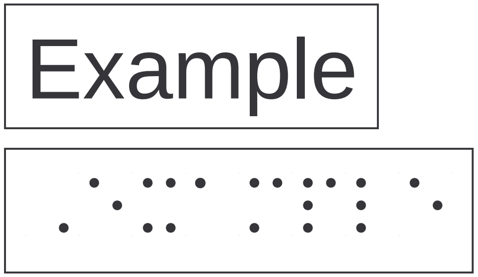
OCR

Pentium III
1600
1500
1400
1.58x per year
1300
1200
1100
1000
HP
9000
900
Relative
800
performance
700
DEC
600
Alpha
500
400
300
1.35x per year
DEC
IBM
HP
MIPS
Alpha
200
Power!
9000
R2000
100
0
#
^
#
^
f
f
f
f
Year
© 2003 Elsevier Science (USA). All rights reserved.
Intel
/

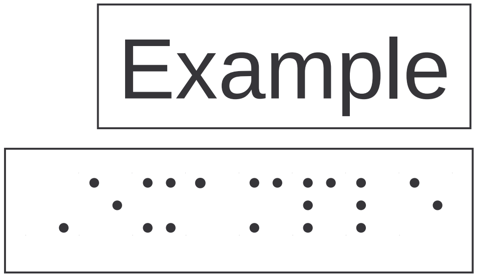
text

Classification of Text Boxes

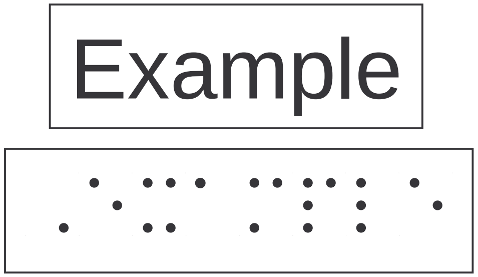
- Text boxes of Braille will be of different size than the original text boxes
  - Mode characters
  - Contractions
  - Braille is fixed width



Left justified

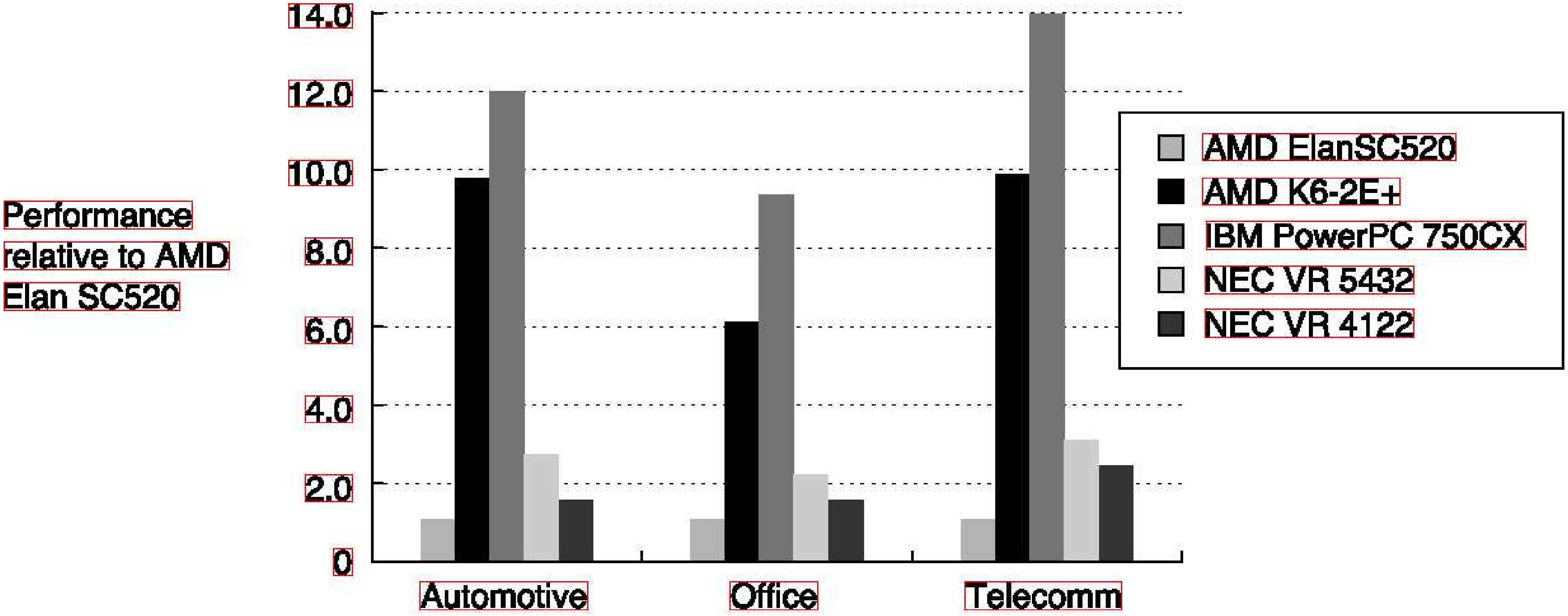


Right justified

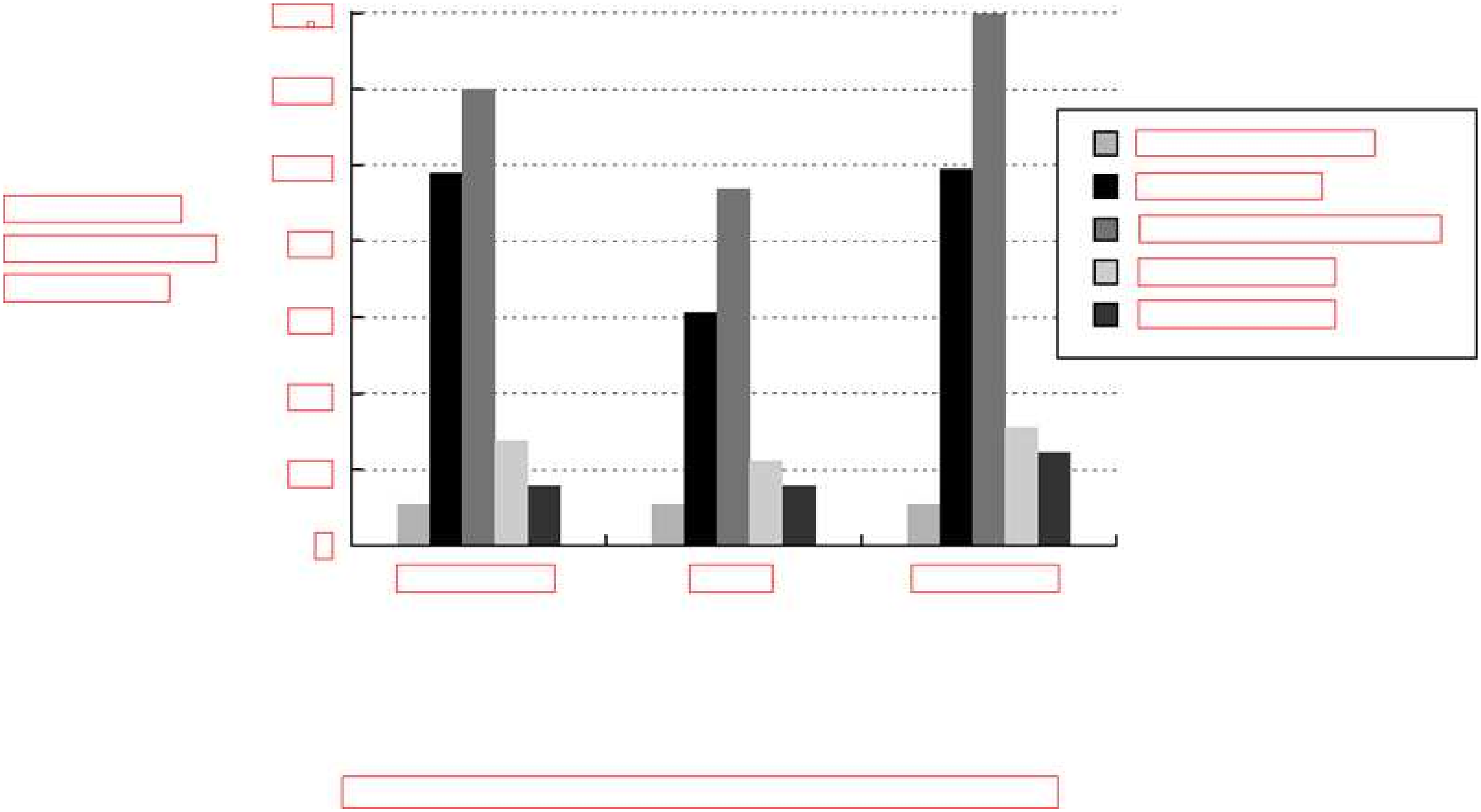


Centered

Perfect Text Boxes



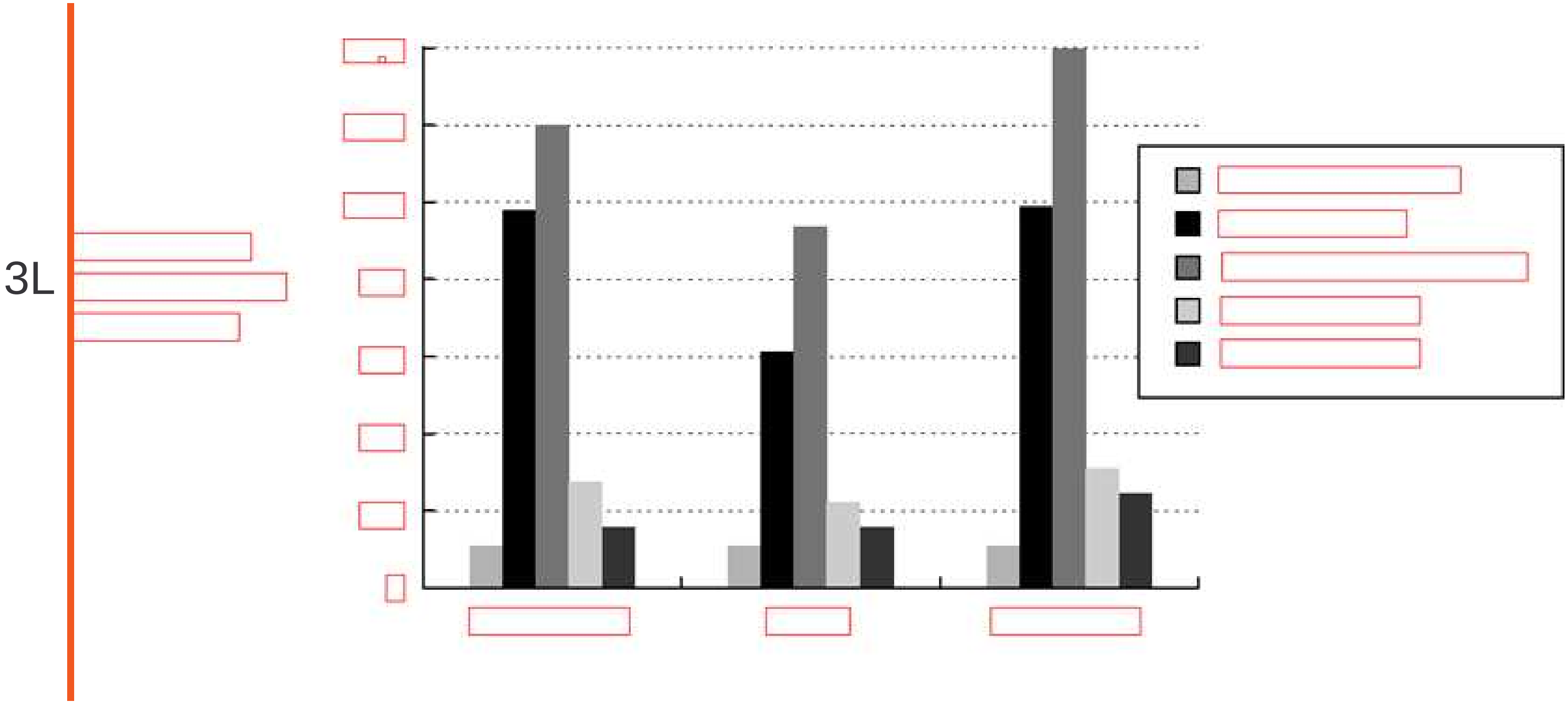
Text Boxes Only



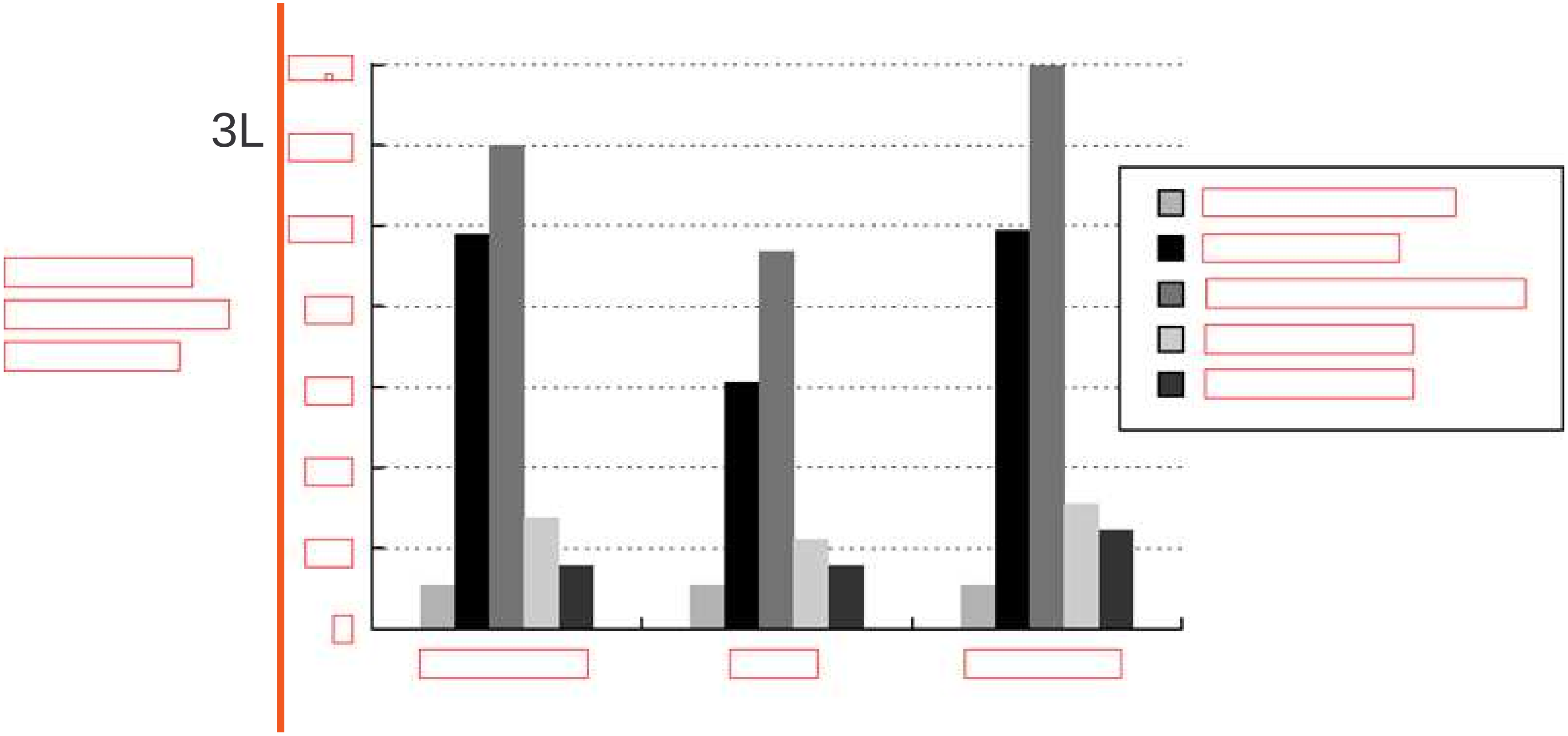
Justification Process

- Sort the upper left and lower right points of text boxes first by x then by y. Use a plane sweep algorithm.
- Left justify - runs (in y) of text boxes with the same (or similar) left x coordinates.
- Right justify - runs (in y) of text boxes with the same (or similar) right x coordinates.
- Center - otherwise

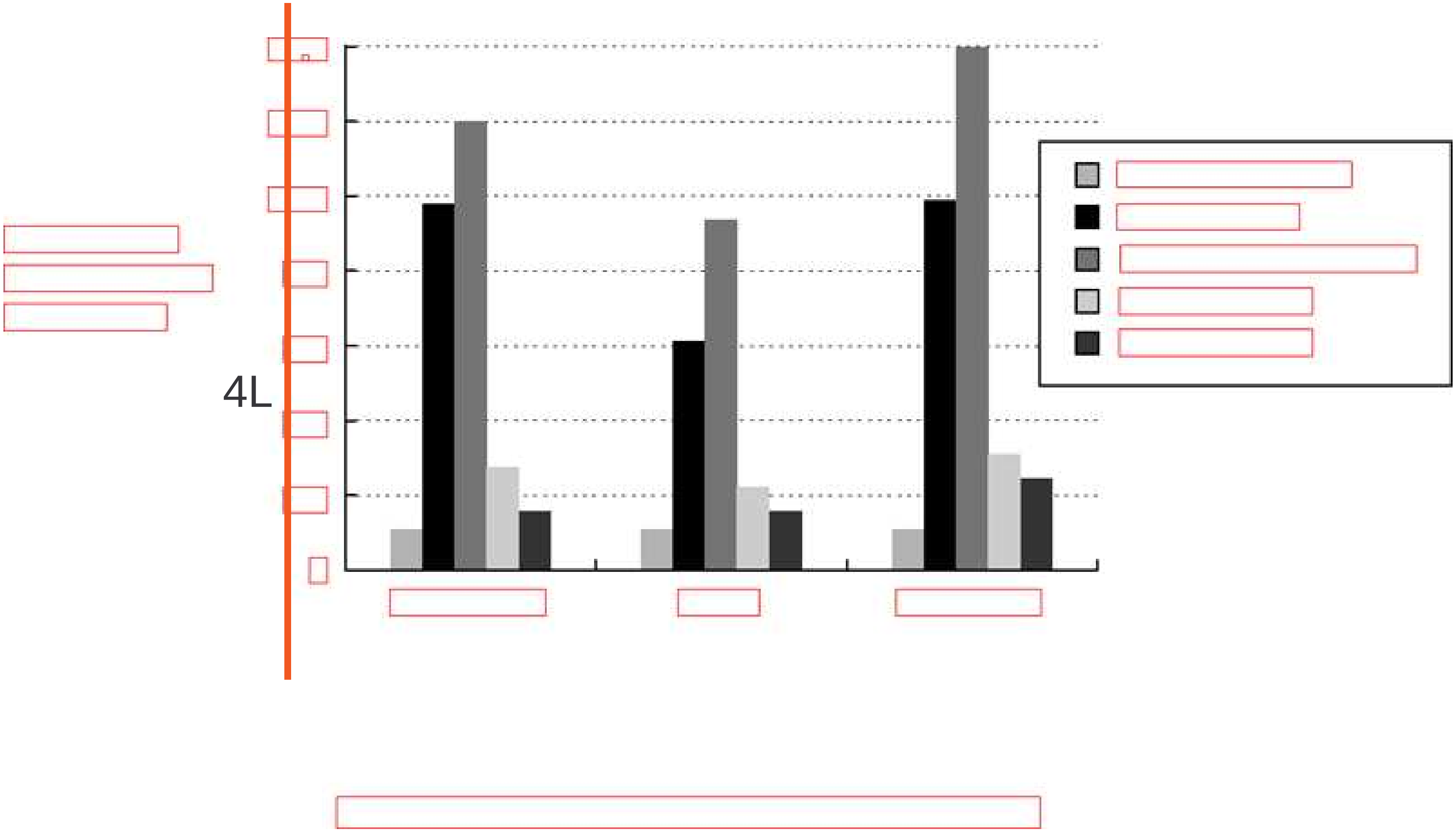
Example Plane Sweep



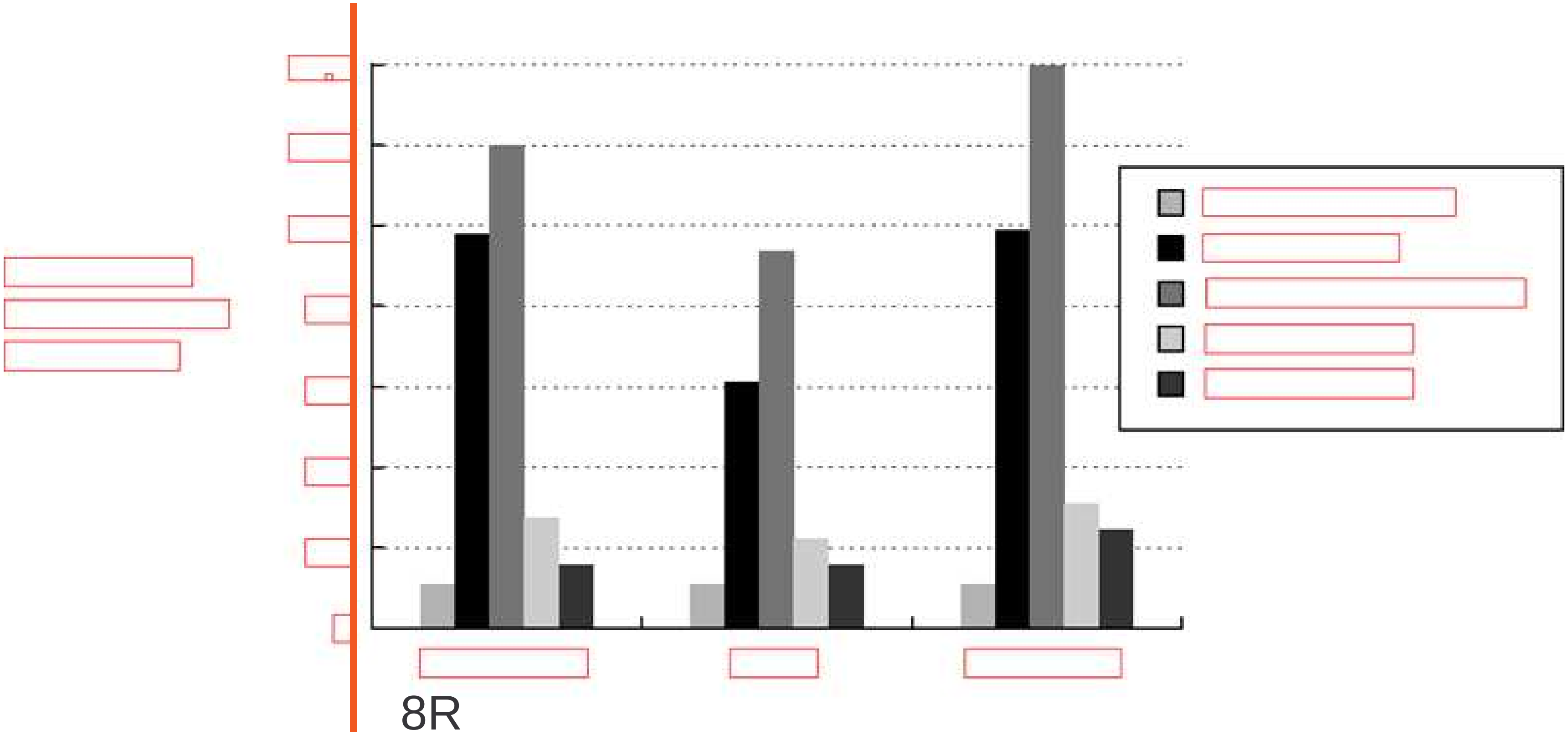
Example Plane Sweep



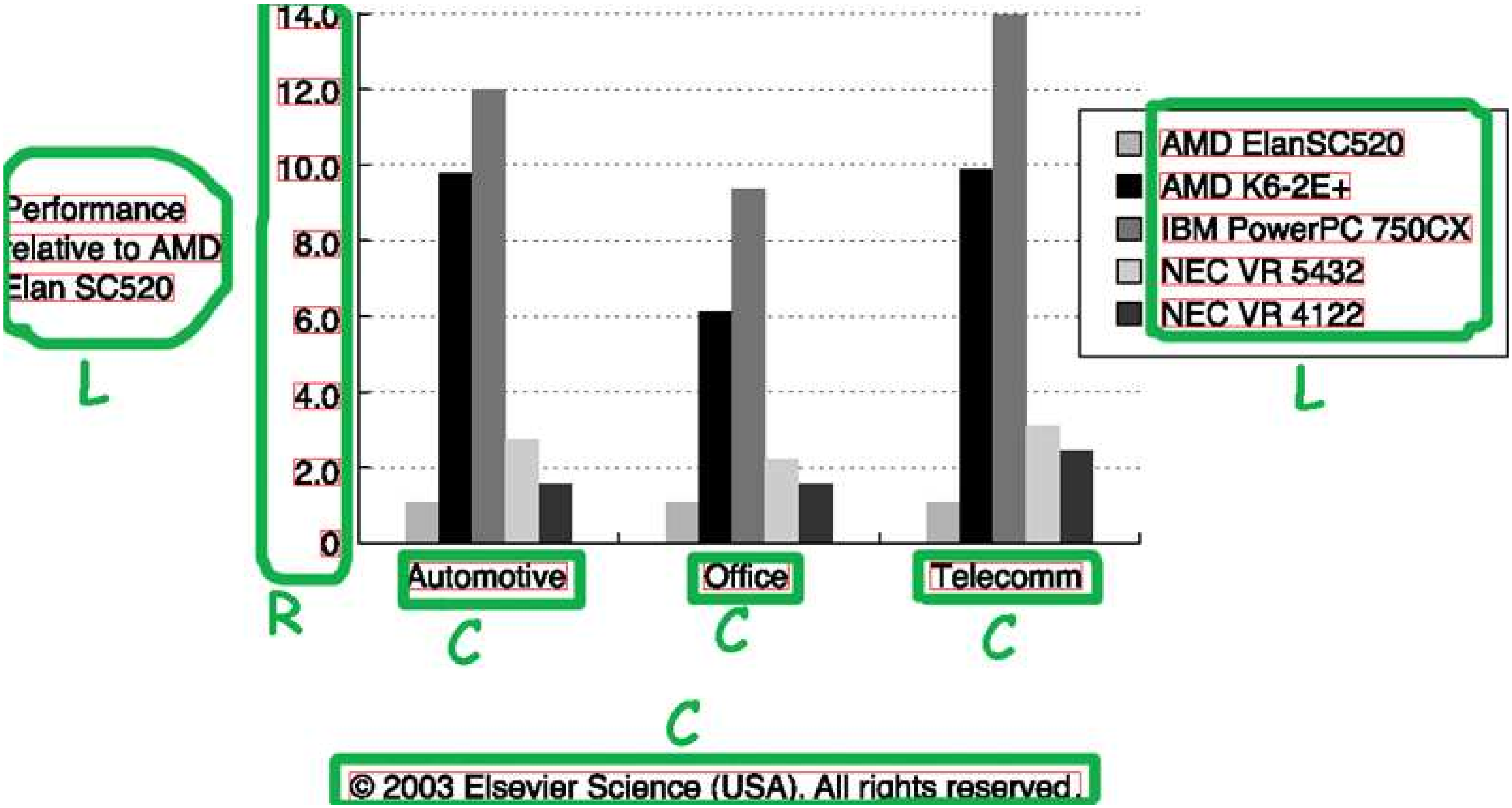
Example Plane Sweep



Example Plane Sweep



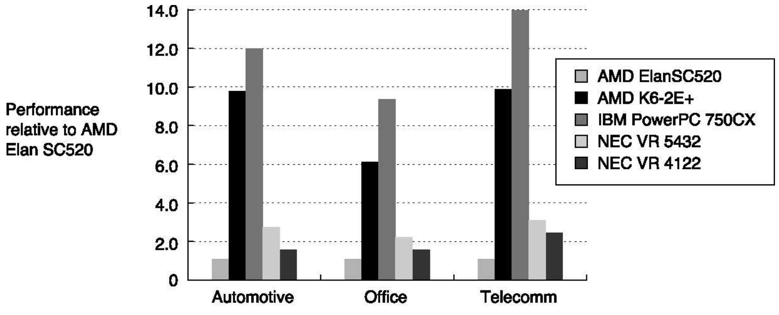
Classification

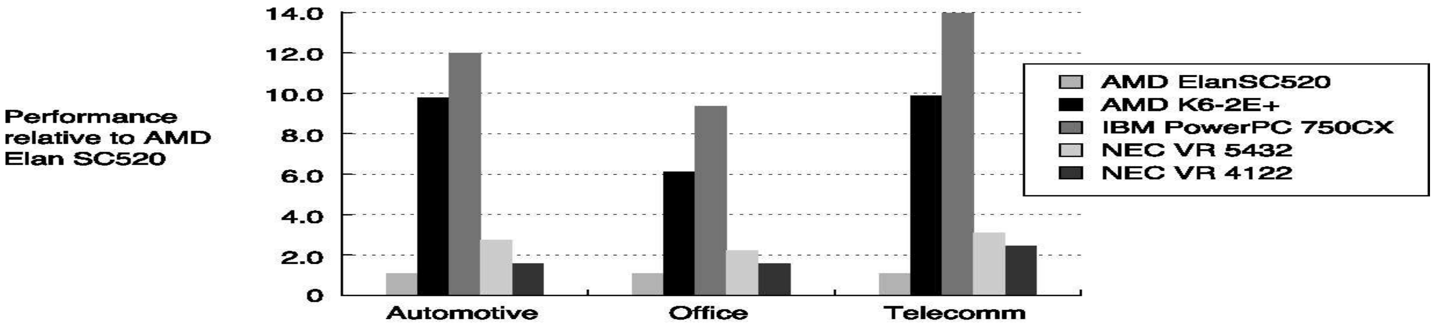


Scaling

- General Procedure
  - Scale in y until the the text height is an acceptable Braille height
  - Scale in x until the Braille correctly justified fits
- The scale factor in x and y may differ, but the distorted image is usually readable.
  - The Braille text is fully readable.
- Scaling procedure is not always successful because of limited paper size.
  - Automatic abbreviations

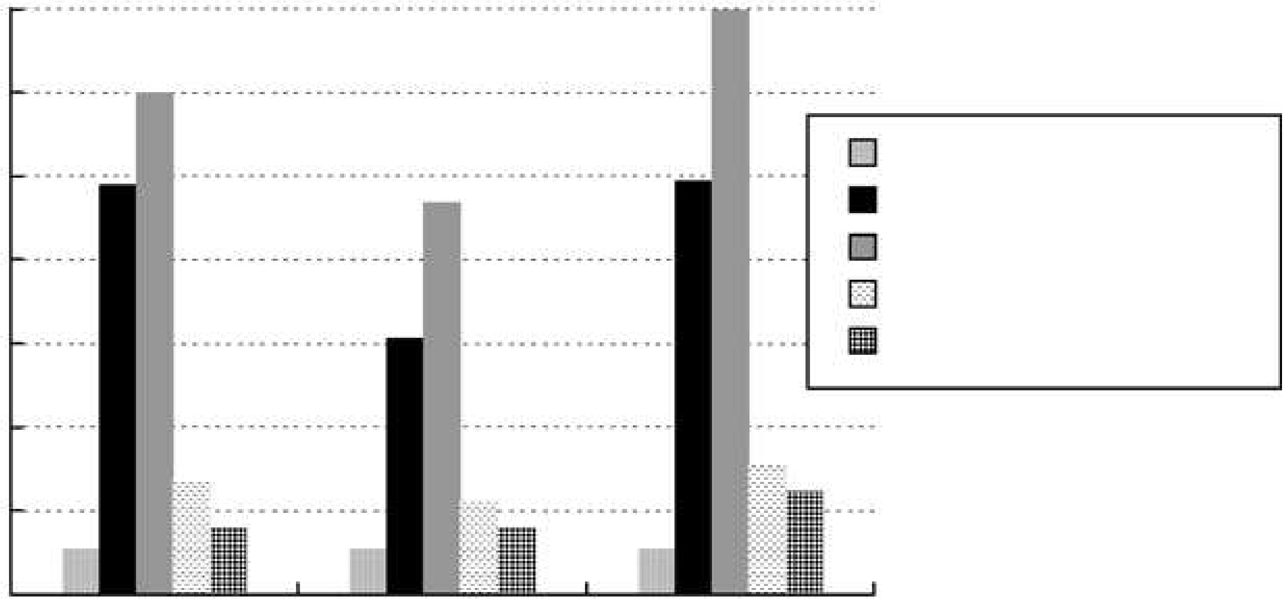
Scaling Example



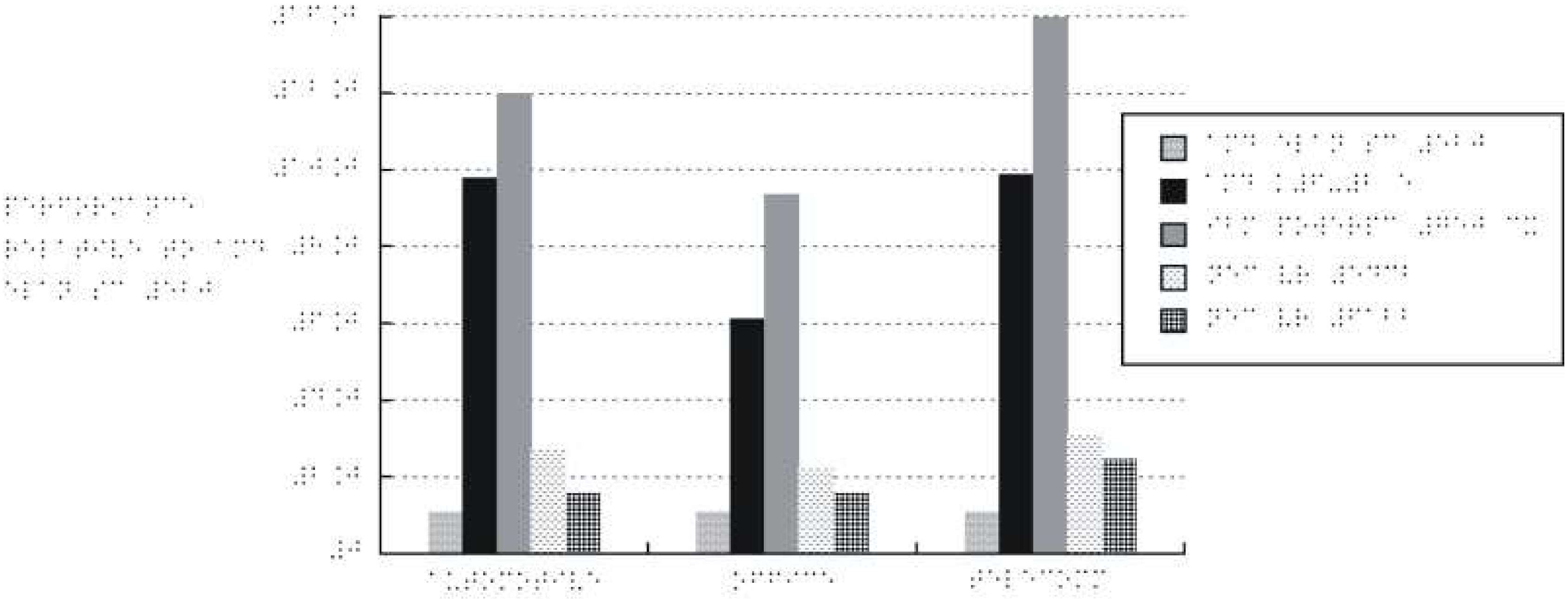


© 2003 Elsevier Science (USA). All rights reserved.

Color Replacement with Texture



Final Result



[illegible]