Approximation Algorithms for Data Placement on Parallel Disks

Srinivas Kashyap
Department of Computer Science
University of Maryland, College Park
raaghav@cs.umd.edu

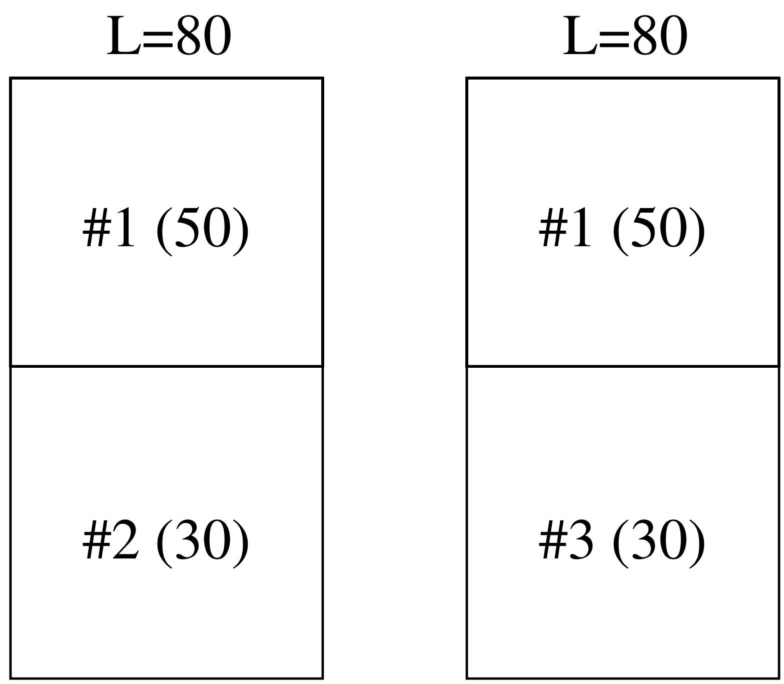
26th March 2003

The Problem

- **Given**:
  - ⋆ M data items. i^{th} data item has size s_i, demand l_i.
  - ⋆ N disks. Each disk has *storage capacity* K and *load capacity* L.

- **Goal**: Find a placement of data items on disks and an assignment of clients to disks to maximize total number of clients served.

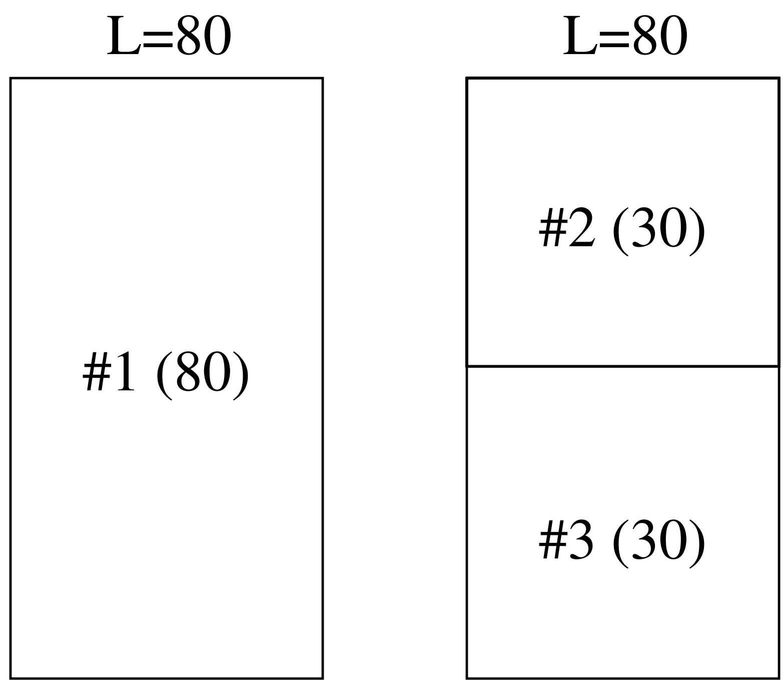
Example (k=2), Optimal Assigment



Item #1: Size = 1, Load = 100
Item #2: Size = 1, Load = 30
Item #3: Size = 1, Load = 30

Fraction Packed = 1

Example (k=2), Non-Optimal Assignment



Item #1: Size = 1, Load = 100

Item #2: Size = 1, Load = 30

Item #3: Size = 1, Load = 30

Fraction Packed = 7 / 8

Related Work

- Class constrained knapsack problem (unit size items) (Shachnai and Tamir)
- Algorithm with tight bound for unit-size items (Golubchik, Khanna, Khuller, Thurimella, Zhu)
- NP-hard for any fixed $k \geq 2$ (Golubchik, Khanna, Khuller, Thurimella, Zhu)

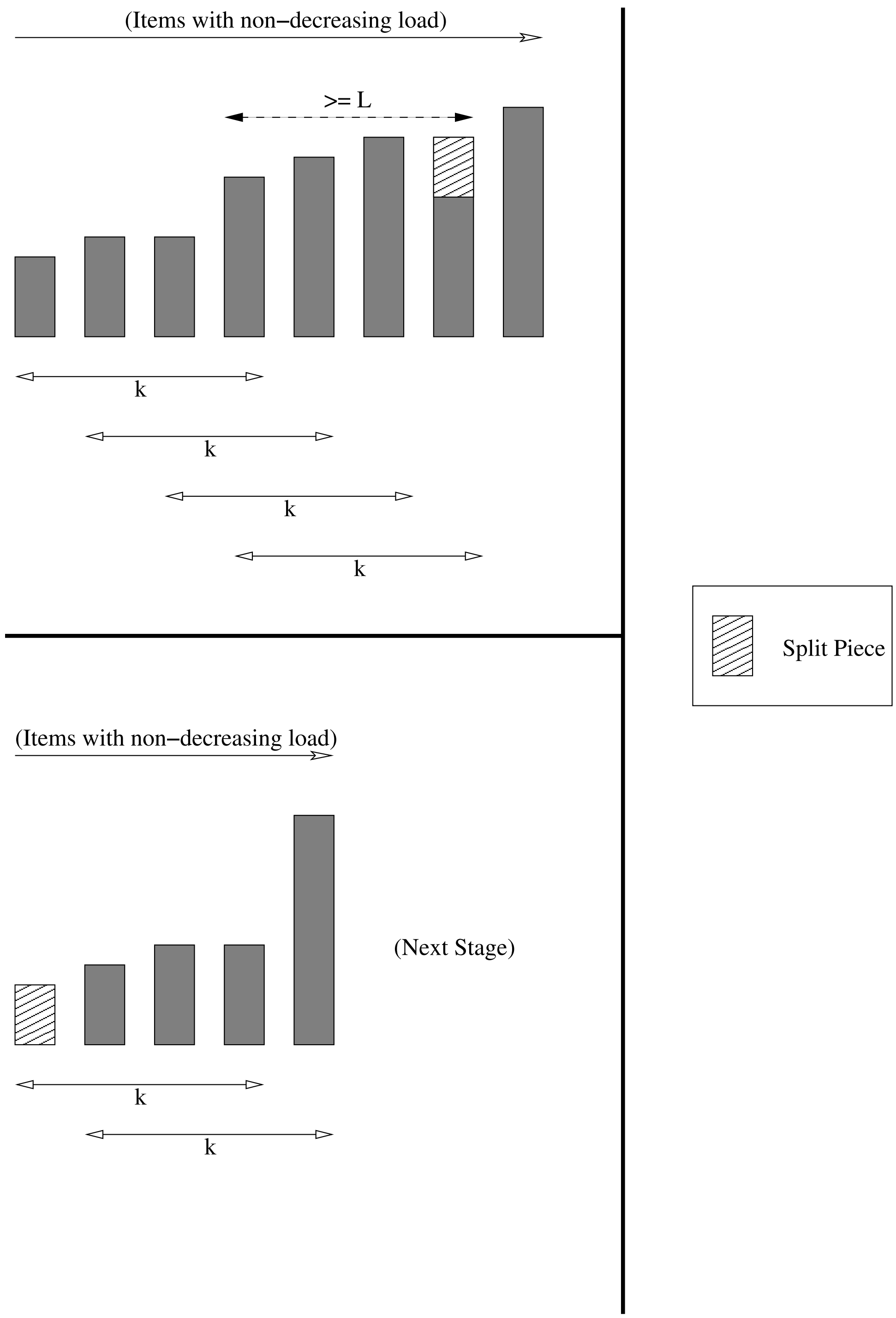
Our Results (Kashyap and Khuller)

- PTAS for arbitrary $s_i \in \{1, \ldots, \Delta\}$. Constant Δ.
- Algorithm with tight bound, when $s_i \in \{1, 2\}$. Cannot guarantee to do better than $(1 - \frac{1}{(1+\sqrt{\lfloor k/2 \rfloor})^2})$ in this case.

Assumptions

- $\sum_{i=1}^{M} l_i \leq N \cdot L$
- $\sum_{i=1}^{M} s_i \leq N \cdot k$

Sliding Window Algorithm (unit size items), k=4

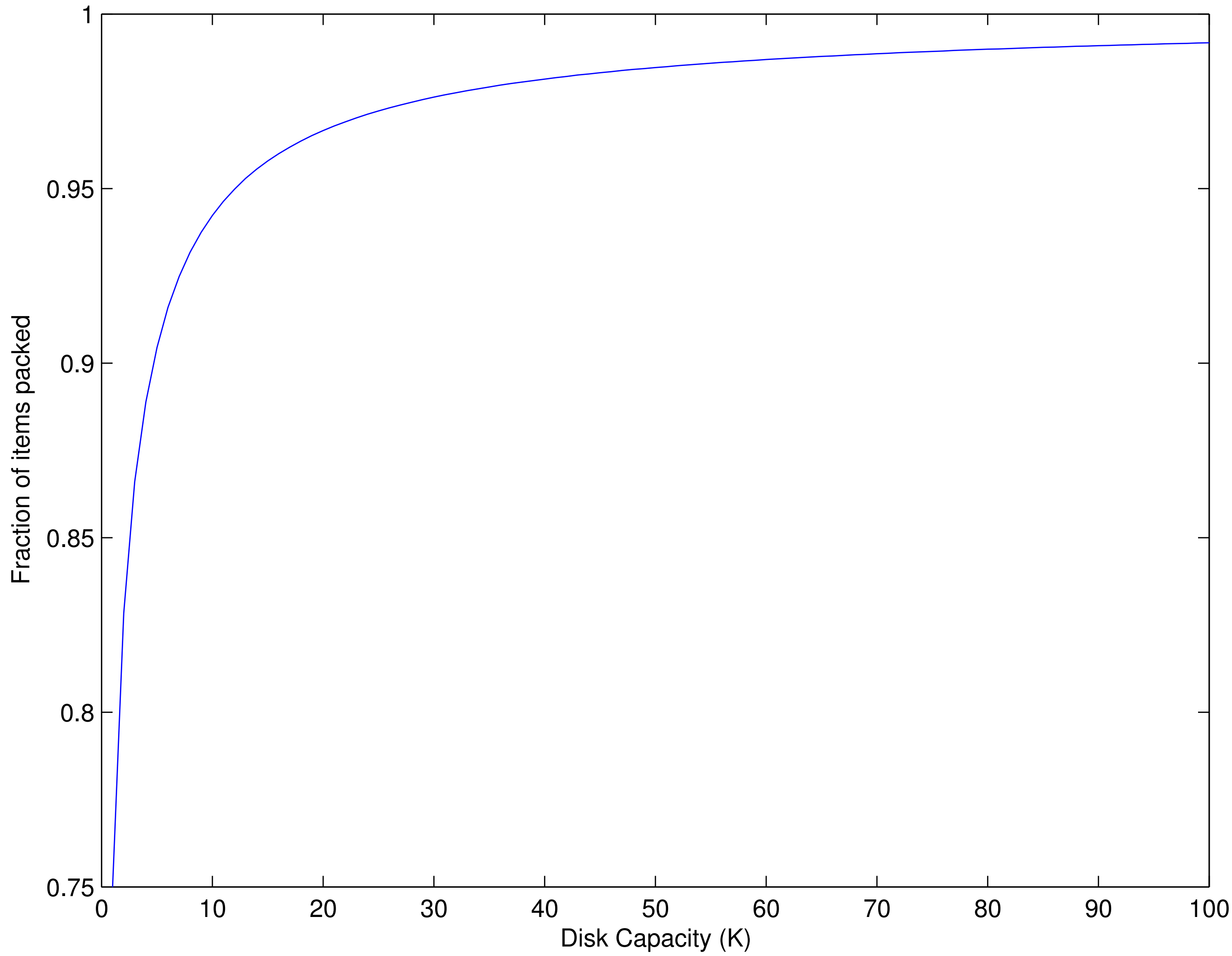


Solution structure(unit size items)

Theorem 1. *It is always possible to pack a* $(1 - \frac{1}{(1+\sqrt{k})^2})$*-fraction of items for any instance.*

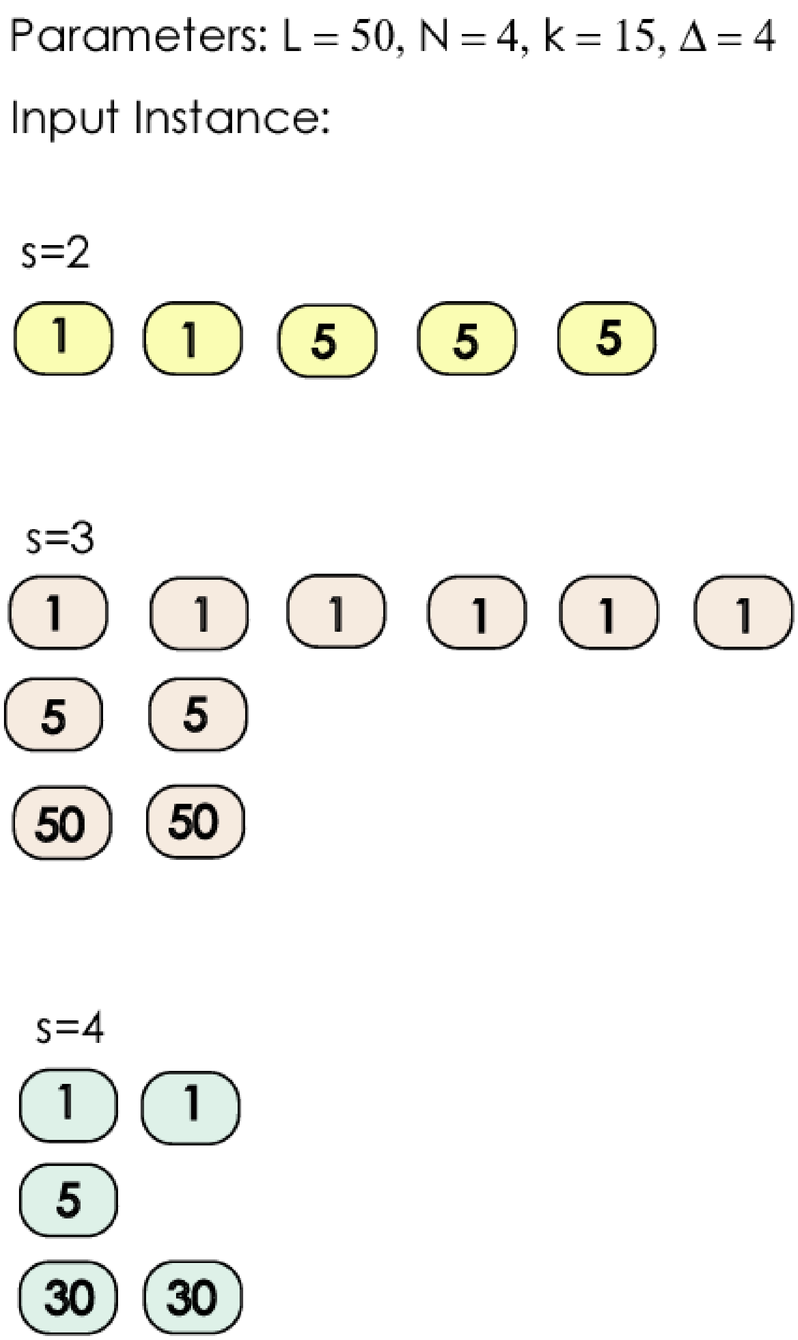
The bound is tight!

The function $(1 - \frac{1}{(1+\sqrt{k})^2})$



The single-list SW algorithm ($s_i \in \{1, \ldots, \Delta\}$)

$$\rho_i = l_i / s_i$$



Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	**= 4, 4, 3, 3, 3**	**(size = 17, load = 5)**
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	**= 4, 3, 3, 3, 3**	**(size = 16, load = 5)**
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	**= 4, 3, 3, 3, 3, 3**	**(size = 19, load = 6)**
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	**= 3, 3, 3, 3, 3, 3**	**(size = 18, load = 6)**
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	**= 3, 3, 3, 3, 3, 2**	**(size = 17, load = 6)**
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	**= 3, 3, 3, 3, 3, 2, 2**	**(size = 19, load = 7)**
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1, 1, 1, 1, 1, (1, 1, 1, 1, 1, 5,) 5, 5, 5, 5, 5, 30, 30, 50, 50
4, 4, 3, 3, 3, (3, 3, 3, 2, 2, 4,) 3, 3, 2, 2, 2, 4, 4, 3, 3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	**= 3, 3, 3, 2, 2, 4**	**(size = 17, load = 10)**
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	**= 3, 3, 2, 2, 4, 3**	**(size = 17, load = 14)**
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 30, 30, 50, 50
4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 4, 3, 3, 2, 2, 2, 4, 4, 3, 3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	**= 3, 2, 2, 4, 3, 3**	**(size = 17, load = 18)**
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 30, 30, 50, 50

4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 4, 3, 3, 2, 2, 2, 4, 4, 3, 3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	**= 3, 2, 2, 4, 3, 3, 2**	**(size = 19, load = 23)**
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1	1	1	1	1	1	1	1	1	1	5	5	5	5	5	5	30	30	50	50
4	4	3	3	3	3	3	3	2	2	4	3	3	2	2	2	4	4	3	3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	**= 2, 2, 4, 3, 3, 2, 2**	**(size = 18, load = 27)**
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 30, 30, 50, 50
4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 4, 3, 3, 2, 2, 2, 4, 4, 3, 3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	**= 2, 4, 3, 3, 2, 2, 2**	**(size = 18, load = 31)**
Sel13	= 3, 3, 2, 2, 2, 4	(size = 16, load = 55)

Phase 1

Remaining Items List:

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, **5, 5, 5, 5, 5, 30**, 30, 50, 50
4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 4, **3, 3, 2, 2, 2, 4**, 4, 3, 3

k+delta = 19

Sel01	= 4, 4, 3, 3, 3	(size = 17, load = 5)
Sel02	= 4, 3, 3, 3, 3	(size = 16, load = 5)
Sel03	= 4, 3, 3, 3, 3, 3	(size = 19, load = 6)
Sel04	= 3, 3, 3, 3, 3, 3	(size = 18, load = 6)
Sel05	= 3, 3, 3, 3, 3, 2	(size = 17, load = 6)
Sel06	= 3, 3, 3, 3, 3, 2, 2	(size = 19, load = 7)
Sel07	= 3, 3, 3, 2, 2, 4	(size = 17, load = 10)
Sel08	= 3, 3, 2, 2, 4, 3	(size = 17, load = 14)
Sel09	= 3, 2, 2, 4, 3, 3	(size = 17, load = 18)
Sel10	= 3, 2, 2, 4, 3, 3, 2	(size = 19, load = 23)
Sel11	= 2, 2, 4, 3, 3, 2, 2	(size = 18, load = 27)
Sel12	= 2, 4, 3, 3, 2, 2, 2	(size = 18, load = 31)
Sel13	**= 3, 3, 2, 2, 2, 4**	**(size = 16, load = 55)**

Phase 1

Disk #1, Selection
5, 5, 5, 5, 5, 25
3, 3, 2, 2, 2, 4 (size = 16, load = 50)
Remaining Items List:
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, **5**, 30, 50, 50
4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 4, **4**, 4, 3, 3

Disk #2, Selection
1, 1, 5, 5, 30, 8
2, 2, 4, 4, 4, 3 (size = 19, load = 50)
Remaining Items List:
1, 1, 1, 1, 1, 1, 1, 1, **42**, 50
4, 4, 3, 3, 3, 3, 3, 3, **3**, 3

Disk #3, Selection
1, 1, 1, 1, 42, 4 (size = 18, load = 50)
3, 3, 3, 3, 3, 3
Remaining Items List:
1, 1, 1, 1, **46**
4, 4, 3, 3, **3**

Disk #4, Selection
1, 1, 1, 1, 46 (size = 17, load = 50)
4, 4, 3, 3, 3
Remaining Items List:
(empty)

Phase 2

Disk #1, Selection
5, 5, 5, 5, 5, 25
3, 3, 2, 2, 2, 4 (size = 13, load = 45)

Disk #2, Selection
1, 1, 5, 5, 30, 8
2, 2, 4, 4, 4, 3 (size = 15, load = 48)

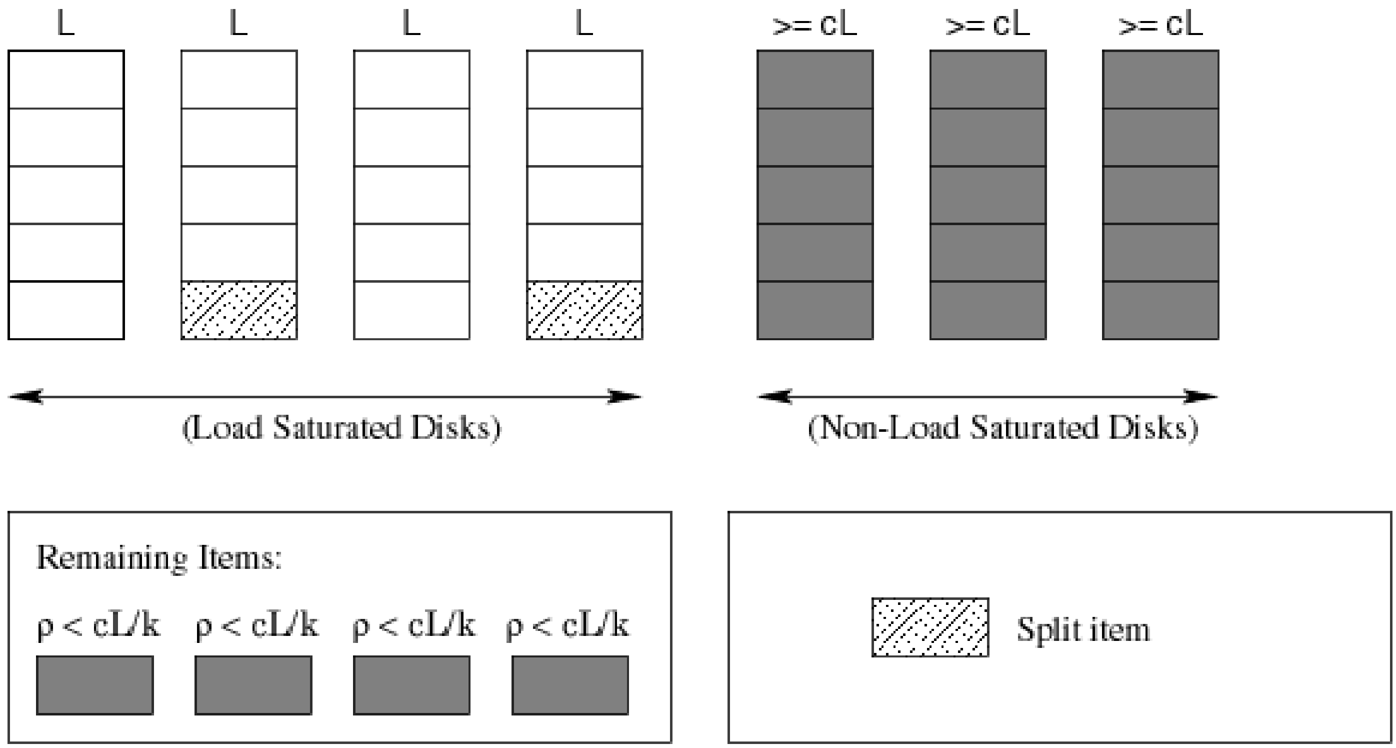
Disk #3, Selection
1, 1, 1, 1, 42, 4 (size = 15, load = 49)
3, 3, 3, 3, 3, 3

Disk #4, Selection
1, 1, 1, 1, 46 (size = 13, load = 49)
4, 4, 3, 3, 3

Remaining Items List:
1, 1, 1, 1, 5
4, 3, 2, 2, 3

Fraction Packed = 0.955

End Phase 1


L
L
L
L
>= cL
>= cL
>= cL
(Load Saturated Disks)
(Non-Load Saturated Disks)
Remaining Items:
ρ < cL/k
ρ < cL/k
ρ < cL/k
ρ < cL/k
Split item

End Phase 1

U: Unassigned load
S: Assigned load
N_l: Load saturated disks
N_s: non Load saturated disks

- $S \geq L \times N_l + c \times N_s \times L$
- $U \leq (1 - c) \times N_s \times L$
- $U \leq \frac{2\Delta N_L c L}{k}$

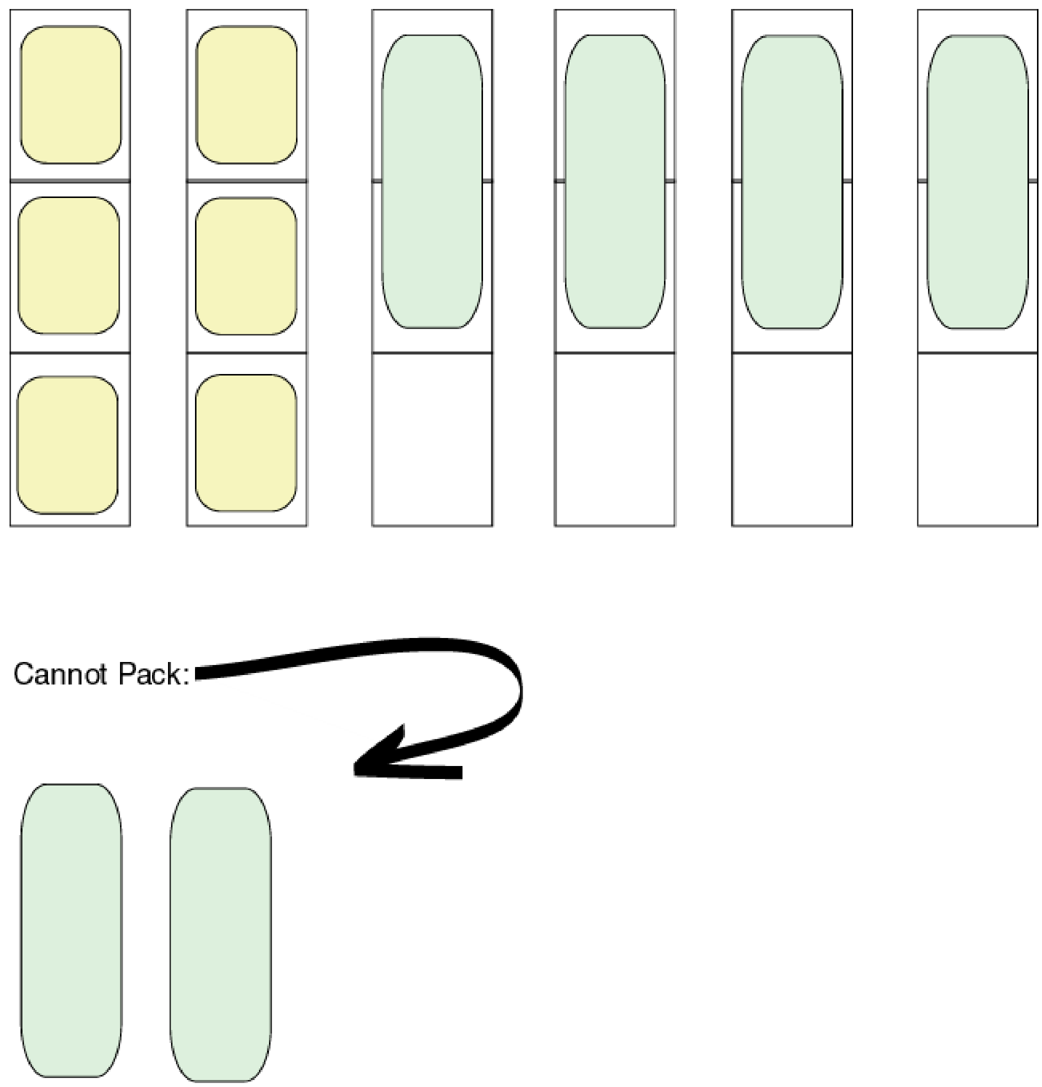
Single-List SW algrotihm

Properties:

- Can guarantee to pack a $1 - \frac{1}{\left(1+\sqrt{\frac{k}{2\Delta}}\right)^2}$ fraction of load by the end of phase-1.
- Lose a $\frac{k-\Delta}{k+\Delta}$ fraction in phase-2.
- Always packs a $\frac{k-\Delta}{k+\Delta}\left(1 - \frac{1}{\left(1+\sqrt{\frac{k}{2\Delta}}\right)^2}\right)$ fraction of load.

The Multi-List SW algorithm ($s_i \in \{1, 2\}$)

Main problem: Fragmentation effect.



Multi-List SW algorithm ($s_i \in \{1, 2\}$)

Algorithm packs a $(1 - \frac{1}{(1+\sqrt{\lfloor k/2 \rfloor})^2})$-fraction of items.
Tight bound.

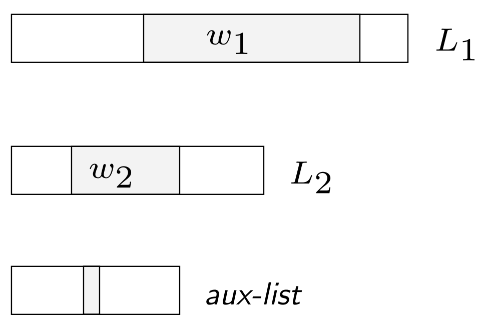
Multi-List SW algorithm ($s_i \in \{1, 2\}$, even K)

- Group size-1 items into groups of size-2. (Will have a size-2 group with a single size-1 item if m_1 is odd)

- Now we can use unit size-SW to solve the problem.

Multi-List SW algorithm ($s_i \in \{1, 2\}$, odd K)

- Maintains three lists:



- L_1 is the list of the first $m_1 - N$ size-1 items (arranged in non-decreasing order of load). If $m_1 < N$, then $L1 = \emptyset$.

- L_2 is the list of size-2 items (arranged in non-decreasing order of load).

- $auxlist$ has the top N (highest demand) size-1 items. [use $N - m_1$ dummy size-1 items if $m_1 < N$]

Multi-List SW algorithm ($s_i \in \{1, 2\}$, odd K)

- $auxlist$ is a "reserve" of size-1 items.
- Forces the selection of an item from $auxlist$ in each disk.

Multi-List SW algorithm ($s_i \in \{1, 2\}$, odd K)

$m_2^{'}$ is the # of size-2 items on the remaining items list.
$m_1^{'}$ is the # of size-1 items.

For each disk, pick and pack the **best** combination of the following selections:

- Select w_2, $0 \leq w_2 \leq \min(\lfloor \frac{k}{2} \rfloor, m_2^{'})$ consecutive size-2 items from L_2 at each of the positions $1 \ldots (m_2^{'} - w_2 + 1)$.

- Select w_1, $0 \leq w_1 \leq \min(k - 2w_2 - 1, m_1^{'})$ consecutive size-1 items from L_1 at each of the positions $1 \ldots (m_1^{'} - w_1 + 1)$

- One size-1 item from $auxlist$ at each of the positions $1 \ldots |$*aux-list*$|$

Multi-List SW algorithm ($s_i \in \{1, 2\}$, odd K)

Picking the **best** combination:

- Let $\mathcal{S}$ be the list of combinations.
- If $\forall s \in \mathcal{S}, load(s) < L$ the algorithm outputs the selection with highest load.
- If $\exists s \in \mathcal{S}$ where $load(s) \geq L$, then let $\mathcal{D}$ be the set of all the selections in $\mathcal{S}$ with load $\geq L$.
- Let $\mathcal{D}^{'} \subseteq \mathcal{D}$ be the set of all the selections which can be made load-feasible by allowing the split of either the highest size-2 item in the selection, or the highest size-1 item (the size-1 item can be either from L_1 or $auxlist$)

Multi-List SW algorithm ($s_i \in \{1,2\}$, odd K)

Picking the **best** combination:

- Define wasted space of a selection to be the sum of the unused space and the size of the item that must be split to make the selection load-feasible.
- Pick the $d \in \mathcal{D}^{'}$ with minimum wasted space.

Reinsert the broken off piece into the appropriate position in the list from which it was picked.

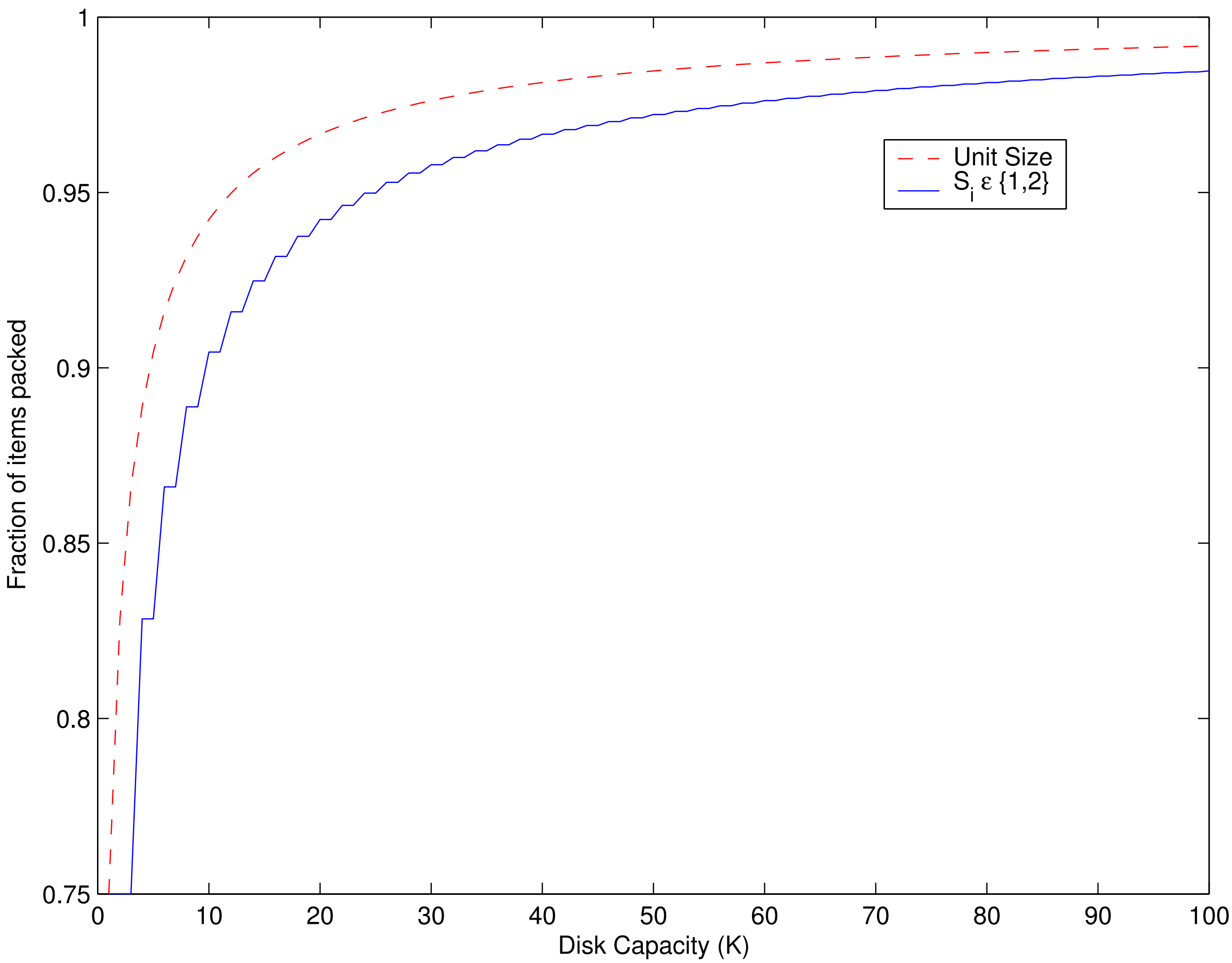
Shrink $auxlist$ if the piece was reinserted in $auxlist$. Move the piece that leaves $auxlist$ into the correct position in L_1.

Solution structure($s_i \in \{1, 2\}$)

Theorem 2. *It is always possible to pack a* $\left(1 - \frac{1}{(1+\sqrt{\lfloor \frac{k}{2} \rfloor})^2}\right)$*-fraction of items for any instance.*

The bound is tight!

Solution structure($s_i \in \{1, 2\}$)



Trivial Tight example, k=3

- Input:

Size-2 items	Load
$N/4$	$5L/2$
$3N/4$	$L/2$

- Optimal Assignment:

Disks	Load
$N/2$	L
$N/2$	$L/2$

Fraction of items packed $= \frac{\frac{NL}{2}+\frac{N}{2}\frac{L}{2}}{NL} = 3/4.$

Polynomial Time Approximation Schemes

- When $(1-\epsilon) > \frac{k-\Delta}{k+\Delta}\left(1 - \frac{1}{\left(1+\sqrt{\frac{k}{2\Delta}}\right)^2}\right)$ (k is a constant). Use another algorithm for constant k and $s_i \in \{a_1, \ldots, a_c\}$.

- Otherwise if $(1-\epsilon) \leq \frac{k-\Delta}{k+\Delta}\left(1 - \frac{1}{\left(1+\sqrt{\frac{k}{2\Delta}}\right)^2}\right)$ use the single-list sliding window algorithm for arbitrary k and $s_i \in \{1, \ldots, \Delta\}$.

Polynomial Time Approximation Schemes

The approximation scheme involves the following basic steps:

1. Any given input instance can be approximated by another instance I' such that no data item in I' has an extremely high demand.

2. For any input instance there exists a near-optimal solution that satisfies certain structural properties concerning how clients are assigned to disks.

3. Finally, we give an algorithm that in polynomial time finds the near-optimal solution referred to in step (2) above, provided the input instance is as determined by step (1) above.

Structured Approximate Solutions

- Disks can be heavy or light. Items can be popular or unpopular
- Heavy disks get all or none of the clients of an unpopular item
- Clients from a popular item **cannot** be distributed over multiple light disks. (consider OPT(I) to be the lexiographically maximal optimal solution)