Blocks World Revisited

John Slaney, Sylvie Thiébaux

Definitions:

- **B** : finite set of blocks (including TABLE)
- **S** : support function. Gives the block x is on
- **σ = <B,S>** : part-state

For a, b in B:

  - **On$_\sigma$(a,b)**: a is on b (S(a)=b)
  - **CLEAR$_\sigma$(a)**: a=table or no block is on a
  - **ABOVE$_\sigma$**: transitive closure of ONσ.
  - **POSITION$_\sigma$(a)**: sequence <a::POSITIONσ(S(a))>

- **Tower**: POSITION of clear block
- **Grounded tower**: ends with table

Definitions:

BW planning **problem**: $\langle \langle \underbrace{B, S_1}_{I} \rangle, \langle \underbrace{B, S_2}_{G} \rangle \rangle$

Move in state σ = <B,S>:
pair of blocks m = <a,b>,

- both CLEAR
- put a on top of b

Plan: sequence of moves $\langle m_1, \ldots m_p \rangle$
that leads from initial state to goal state
Optimal plan: there is no shorter plan

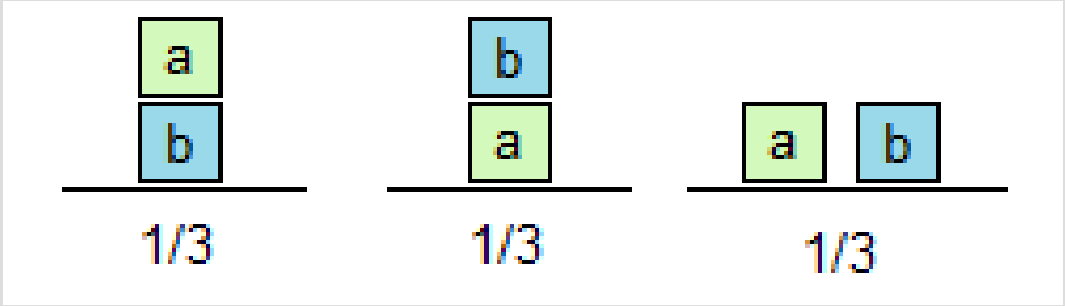
Random BW States

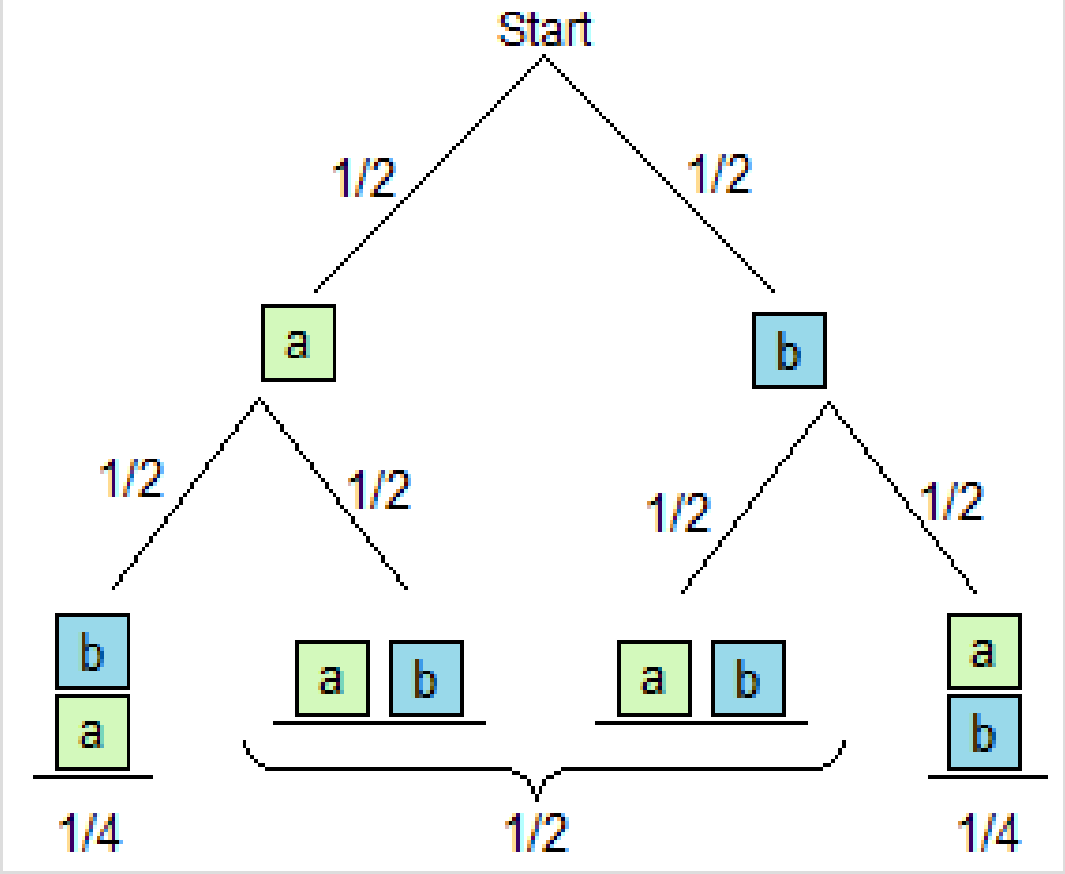
Naive:

- Pick random unplaced block
- Pick random position where to put it

Placing on table should not be as likely as placing on another block!

Random BW States





The States of Blocks World:
Number of states

Needed for:

- Generating representative random states
- Finding out how good plans are

$$g(n,k) = \text{states with } k \text{ grounded, } n \text{ ungrounded towers}$$

Recursive definition:

$$g(n+1,k) = \underbrace{g(n,k+1)}_{\text{put on table}} + \underbrace{(n+k)\,g(n,k)}_{\text{put on other block}}$$

Iterative definition:

$$g(n,k) = \sum_{i=0}^{n} \underbrace{\binom{n}{i}}_{\text{chose } i \text{ towers on table}} \cdot \underbrace{\frac{(n+k-1)!}{(i+k-1)!}}_{\text{Remaining } n-i\text{: put on which tower}}$$

Random BW States:
BWSTATES algorithm

$$R(\phi,\tau)=\frac{g(\phi,)}{g(\phi-1,\tau)}$$

$$\tau = towers\ grounded\ at\ this\ stage$$

1. Start with empty table + n ungrounded single-block towers

2. Repeat until all towers are grounded:

- Arbitrarily select one of the ϕ ungrounded towers
- Using ratio R, select table or another tower

Blocks World Planning:
Definitions

Misplaced: block is not in goal state.
In position: block is in goal state.

Constructive move: moves block into its goal position.

Problem is **deadlocked** if no constructive move is possible. (For any misplaced block b_1, some other block b_2 must be moved before a constructive move is possible.)

Blocks World Planning:
Definitions

Deadlock:

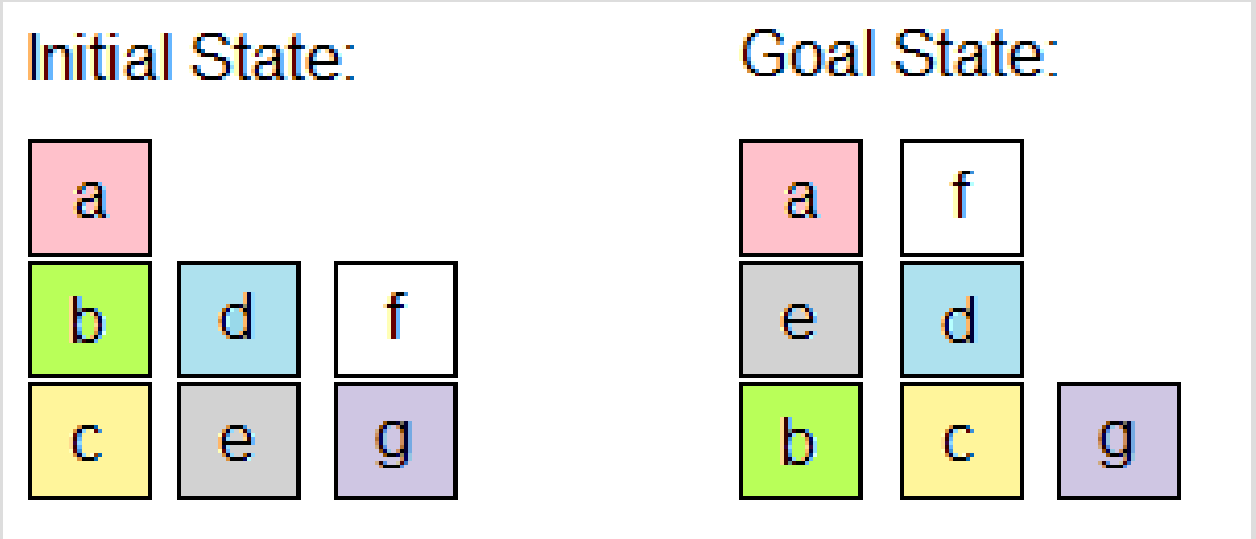
$N_{(I,G)}(a,b) \equiv$

$\text{POSITION}_I(a) \neq \text{POSITION}_G(a) \wedge$

$\text{POSITION}_I(b) \neq \text{POSITION}_G(b) \wedge$

$\exists x \neq \text{TABLE} (\text{ABOVE}_I(b,x) \wedge \text{ABOVE}_G(a,x))$

Example:
a and d



Blocks World Planning:
Breaking deadlocks

At least one block in each deadlock must be moved **twice**. First to table, then to its goal position.

The set of blocks moved to table must be a **hitting set** for the deadlocks, and it should be **minimal**.

This is what makes optimal BW planning difficult!

Strategies for near-optimal BW-planning

US (Unstack-Stack) :

1. Put all misplaced blocks on table
2. Then build goal.

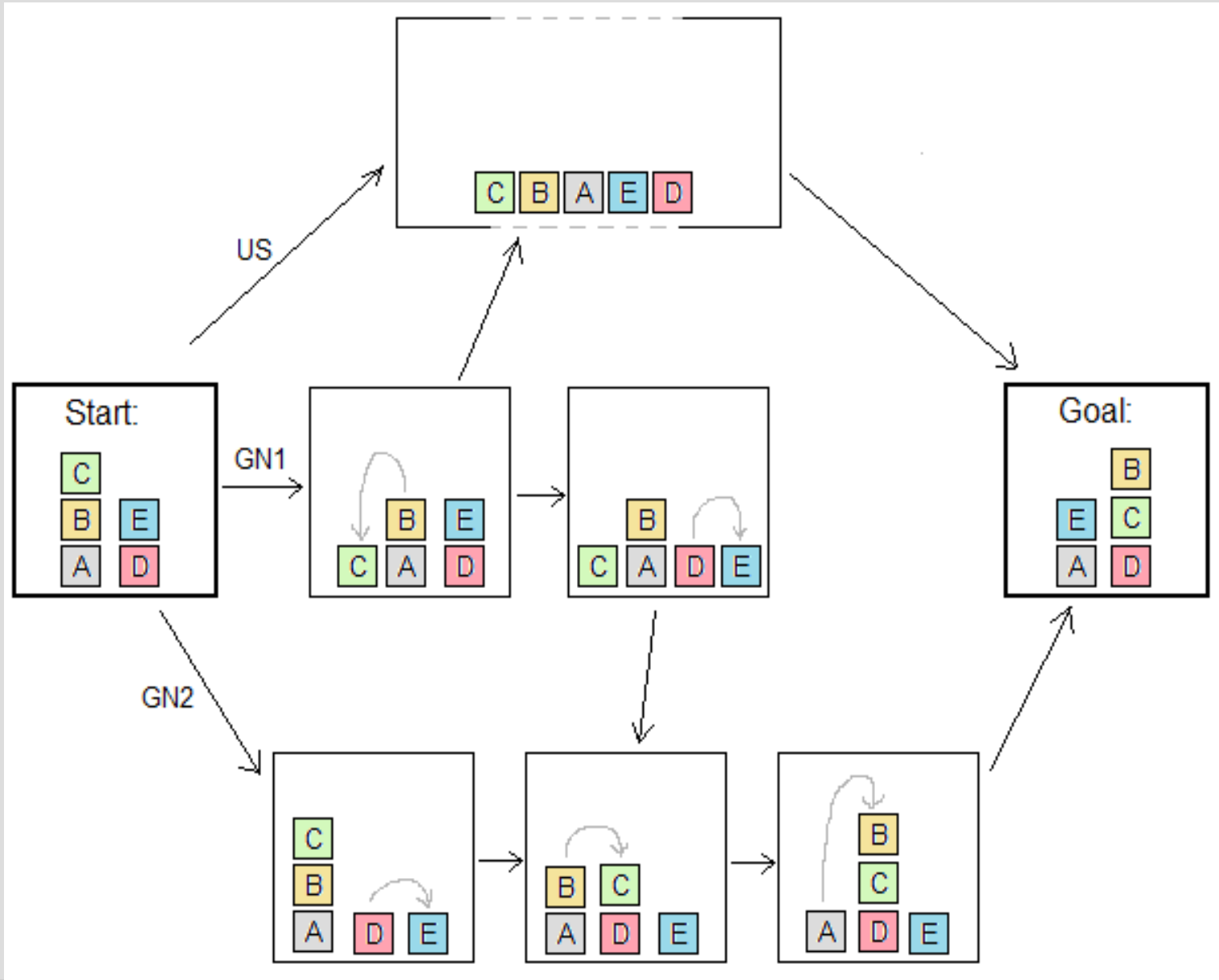
GN1 (Gupta and Nau) :

1. If all blocks in position, stop
2. Perform constructive move <a,b> if one exists
3. Else arbitrarily choose a misplaced clear block and move it on the table.

GN2 :

Same as GN1, but move a deadlocked block

BW-Planning Strategies in comparison



Linear-time algorithm for US

Trick: Find which blocks are in position in O(n), and only once.

Store the values for each parameter, then read them from memory instead of recalculating them recursively.

Similar for STACK and UNSTACK: update stored information by MOVE.

Linear time algorithm for GN1

Status of a block at certain time:

1. READY to move constructively (misplaced, clear; target positioned and clear)
2. STUCK on tower (misplaced, clear, not on table, can't move constructively because target misplaced/not clear
3. OTHER: Neither of the above

Doubly linked list of READY and STUCK blocks (update in constant time)

Linear time algorithm for GN1

- First move READY blocks
- Then move STUCK blocks
- If both lists empty, goal reached

When a block moves, change status and position, as well as:

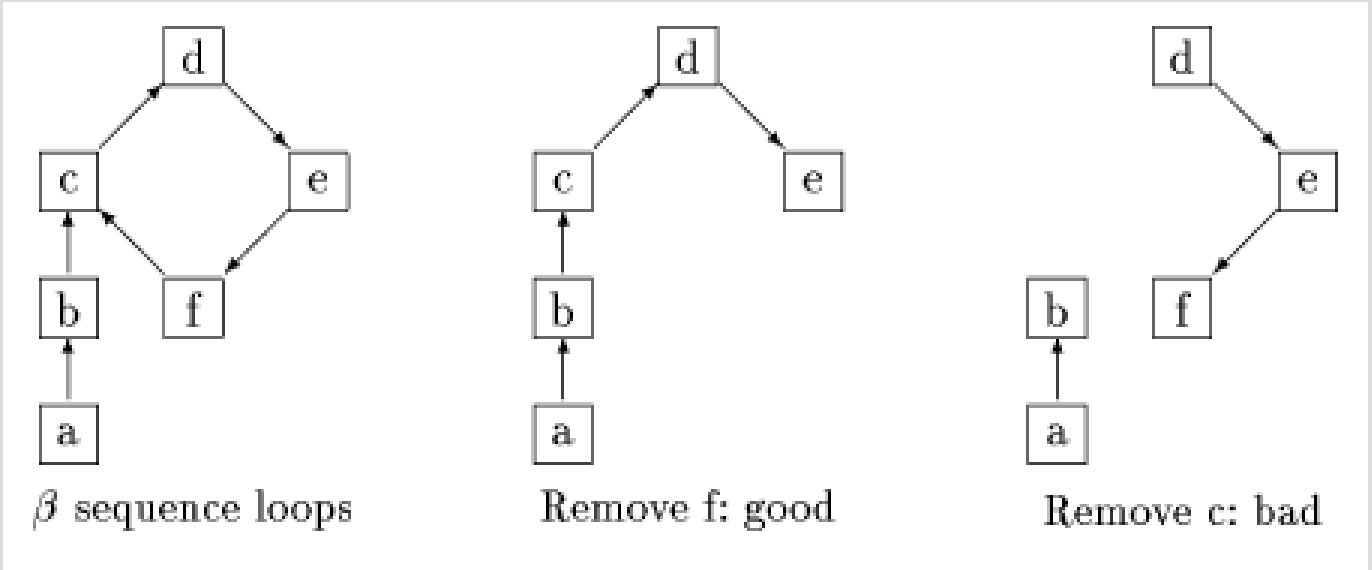
- Blocks currently below A
- Blocks that will be on A in goal
- Blocks which in goal will be on block currently below A

(constant time, since at most 4 blocks can change status)

Linear time algorithm for GN2

- Store the first block of a tower („concierge“)
- Let it keep track of relevant tower information.
- Each block knows its concierge.

Breaking deadlocks: start with the last block before the sequence loops



β sequence loops | Remove f: good | Remove c: bad

Algorithm for optimal BW-planning

Algorithm: PERFECT

```
Construct set K of known deadlocks
(First only singleton deadlocks known)
Loop until finished:
    Generate minimal size hitting set H for K
    Test(H)
    if H solves the problem then
        return Plan(H)
    else
        find a deadlock not in H, and add it to K
```

Algorithm for optimal BW-planning

Requires five procedures to solve sub-problems:

1. Find set of singleton deadlocks to initialise K
2. Generate minimal size H
3. Test H
4. Find new deadlock it doesn't cover
5. Use H to produce a plan

GN1H:

- Modified version of GN1, given a set H of blocks.
- In step 3, only pick blocks in H to break deadlocks.

Finding new deadlocks for GN1H

For each block b, do:
    if H ∪ {b} is not a hitting set then
        add b to H

At the end:

- H is a non hitting set,
- Every proper superset of it is a hitting set
- Its complement in respect to B\{TABLE} is a new deadlock

Finding new deadlocks for GN1H: Example

Problem has deadlocks:
{A, B}, {B}, {C, D}, {D, E}

H ∪ {A} = {A,B} is no hitting set →add A
H ∪ {B} = {A,B} (no need to test)
H ∪ {C} = {A,B,C} is no hitting set →add C
H ∪ {D} = {A,B,C,D} is hitting set →don't add
H ∪ {E} = {A,B,C,E} is hitting set →don't add

Complement {D,E} is a new deadlock.

Experimental observations:
Runtimes

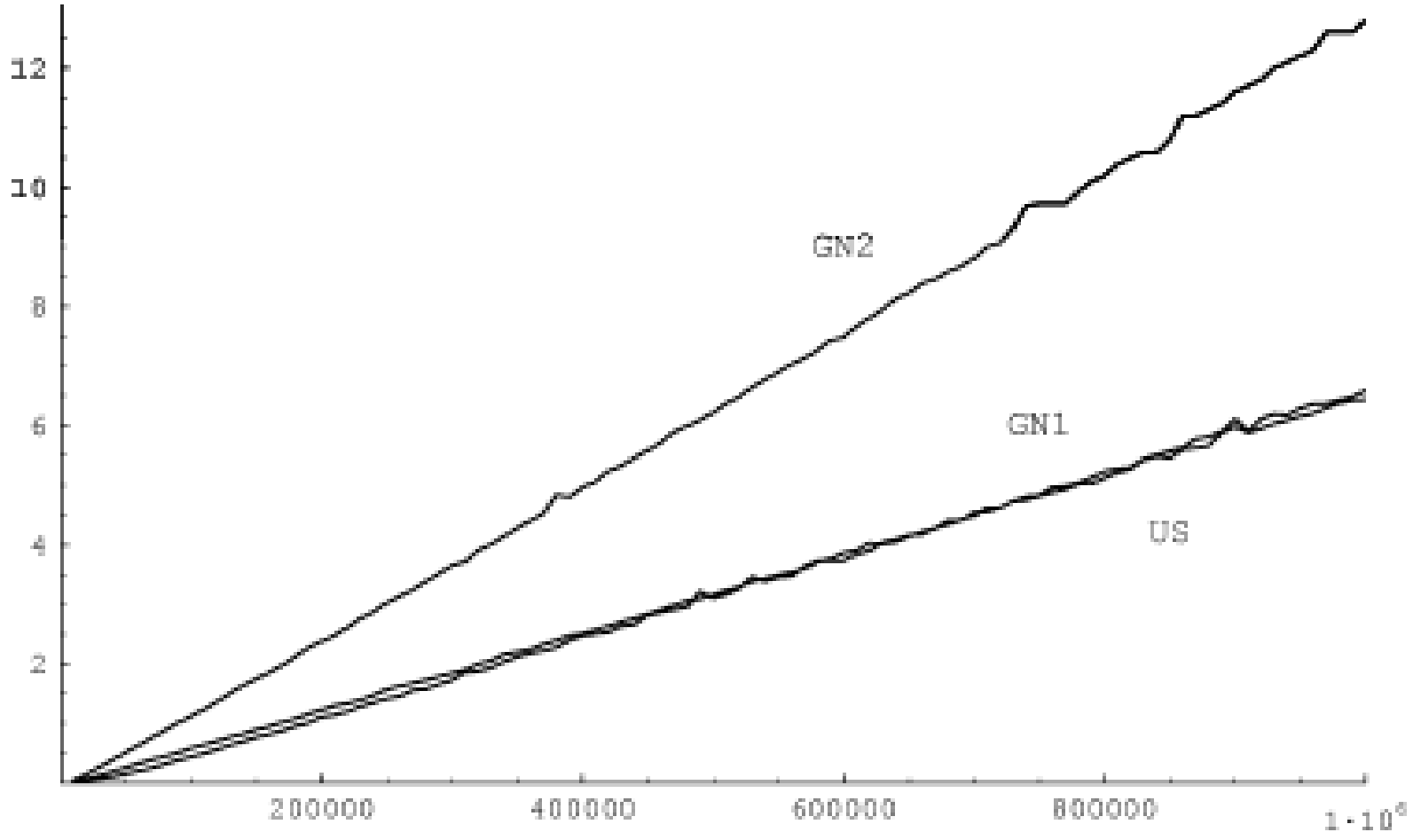


Fig. 8. Average runtimes (in secs) as a function of n for the near-optimal algorithms

Experimental observations:
Average plan length

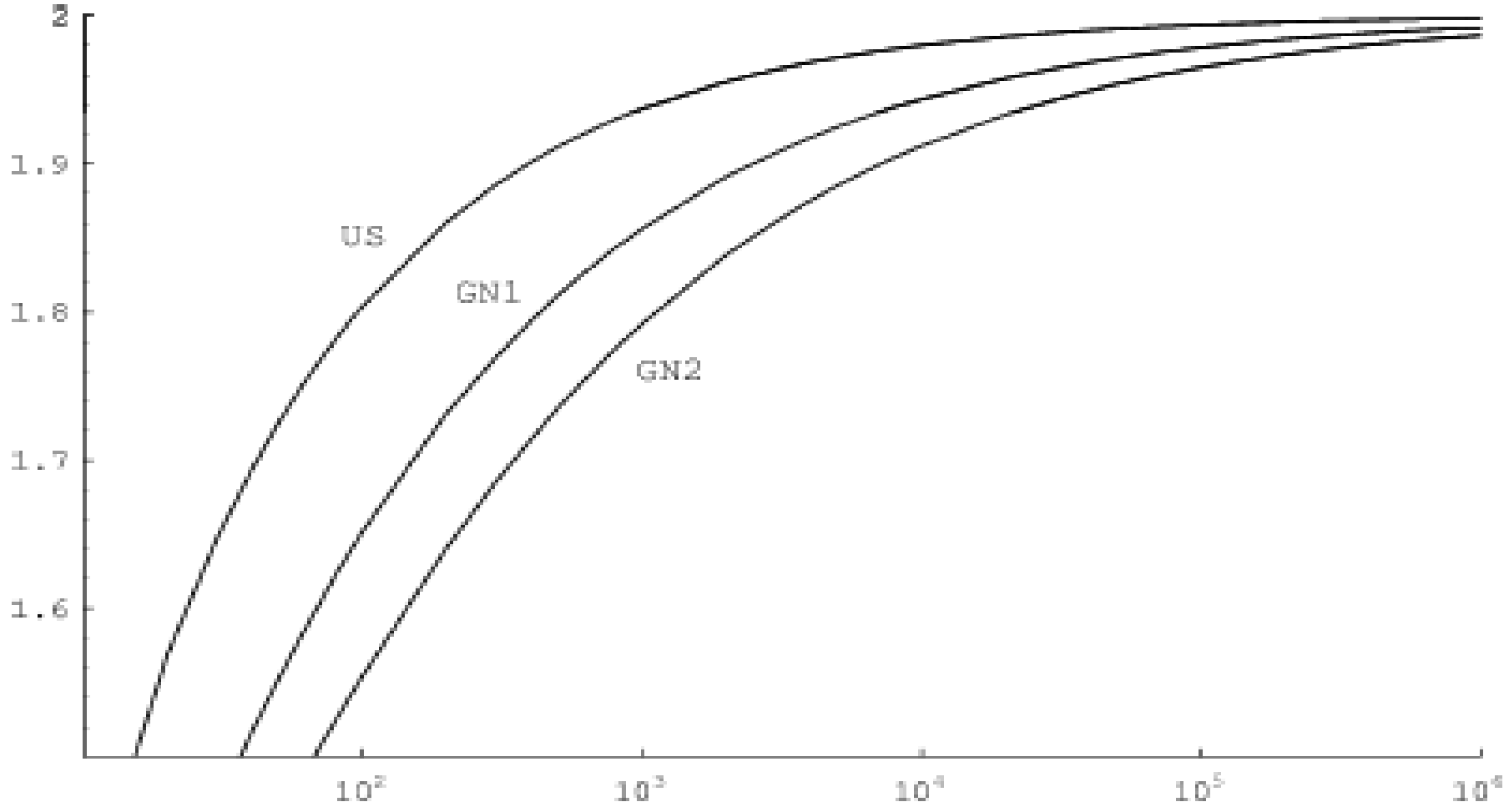


Fig. 11. Average plan length as a function of n: (number of moves)/ n.

Experimental Observations:
Average performance ratios

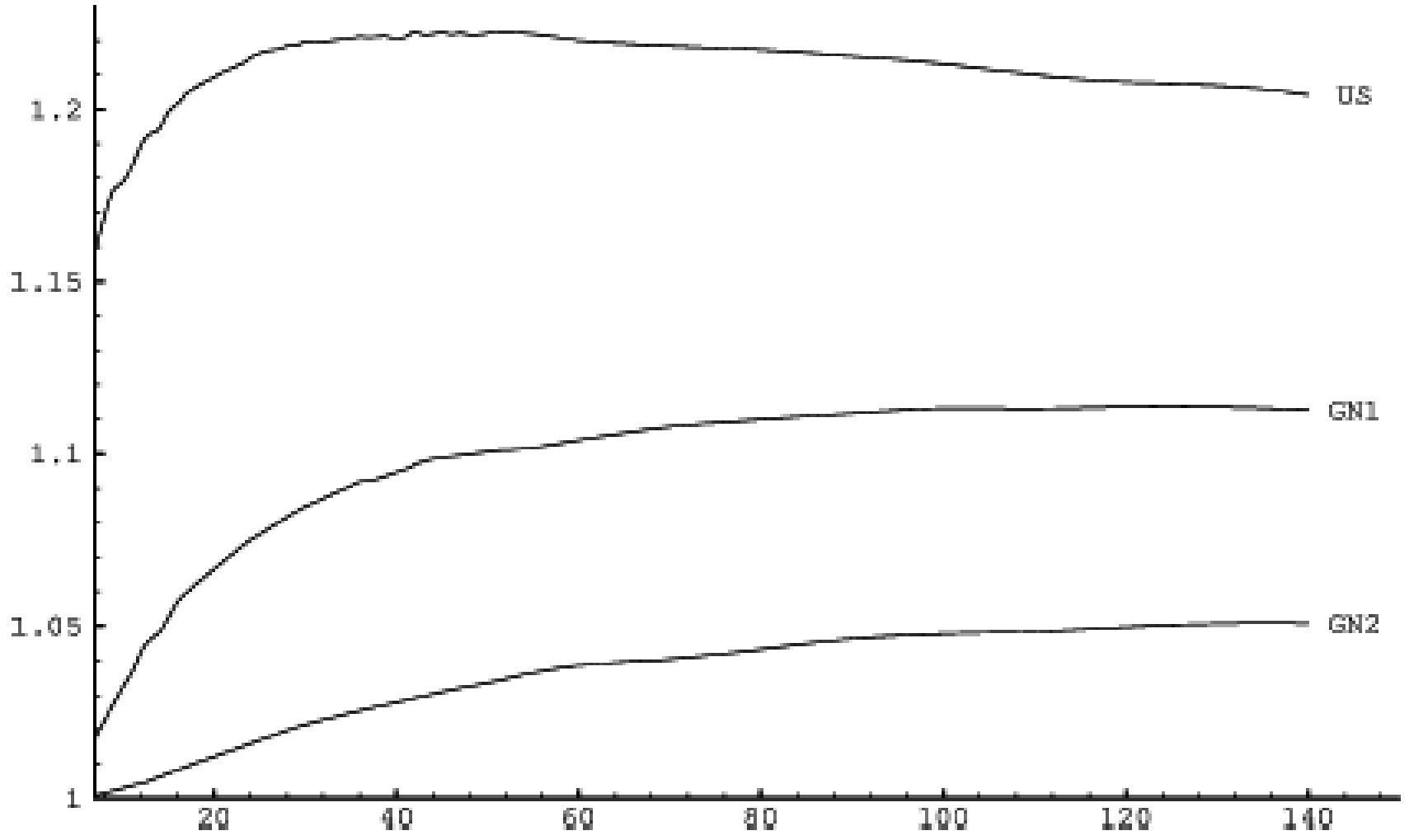


Fig. 13. Average performance ratio as a function of n: (plan length)/(optimal plan length).

Structure of BW planning problems

- Most blocks usually displaced
  →lower bound of roughly n for solution length
- About $\sqrt{(n)}$ blocks on table and cannot be deadlocked
  →upper bound around $2(n-\sqrt{(n)})$
- Of the others, almost none are dead-lock free
  →ignore this, does not help.
- About 40% of blocks are singleton deadlocks
  →Relevant for GN2 which searches for deadlocks

Structure of BW planning problems

Easy problem:

- Few deadlocks (many small towers, under-constrained)
- Small deadlocks (few tall towers, over-constrained)

Hard problem:

- Inbetween, around 14 towers (10-20) for 100 blocks

Most random problems fall into hard region, around 10 towers (for 100 blocks).

Relevant features for planners

- Make constructive moves when you can
- Recognize and break singleton deadlocks
- Knowing non-constructive moves go to table greatly reduces set of possible plans

Conclusion

Blocks world is the „Hello World“ of planning.

Good planning systems have quadratic or linear performance.
Speed is not useful for comparison, but plan length is.

Use hardest problem instances, from region around 14 towers.
For less focus on finding minimal hitting sets: 10 or 20 towers. Else uniformly distribute.

Questions?