Best-First Heuristic Search for Multi-Core Machines

Ethan Burns[1], Seth Lemons[1], Rong Zhou[2] and Wheeler Ruml[1]

1 University of New Hampshire

2 parc Palo Alto Research Center

[NSF grant IIS-0812141]

Motivation: The Future is Multicore

> Now we're into the explicit parallelism multiprocessor era, and this will dominate for the foreseeable future. I don't see any technology or architectural innovation on the horizon that might be competitive with this approach.

John Hennessy
President of Stanford University,
Cofounder of MIPS Computer Systems

(*A Conversation with John Hennessy and David Patterson*, ACM Queue, December 2006)

Overview

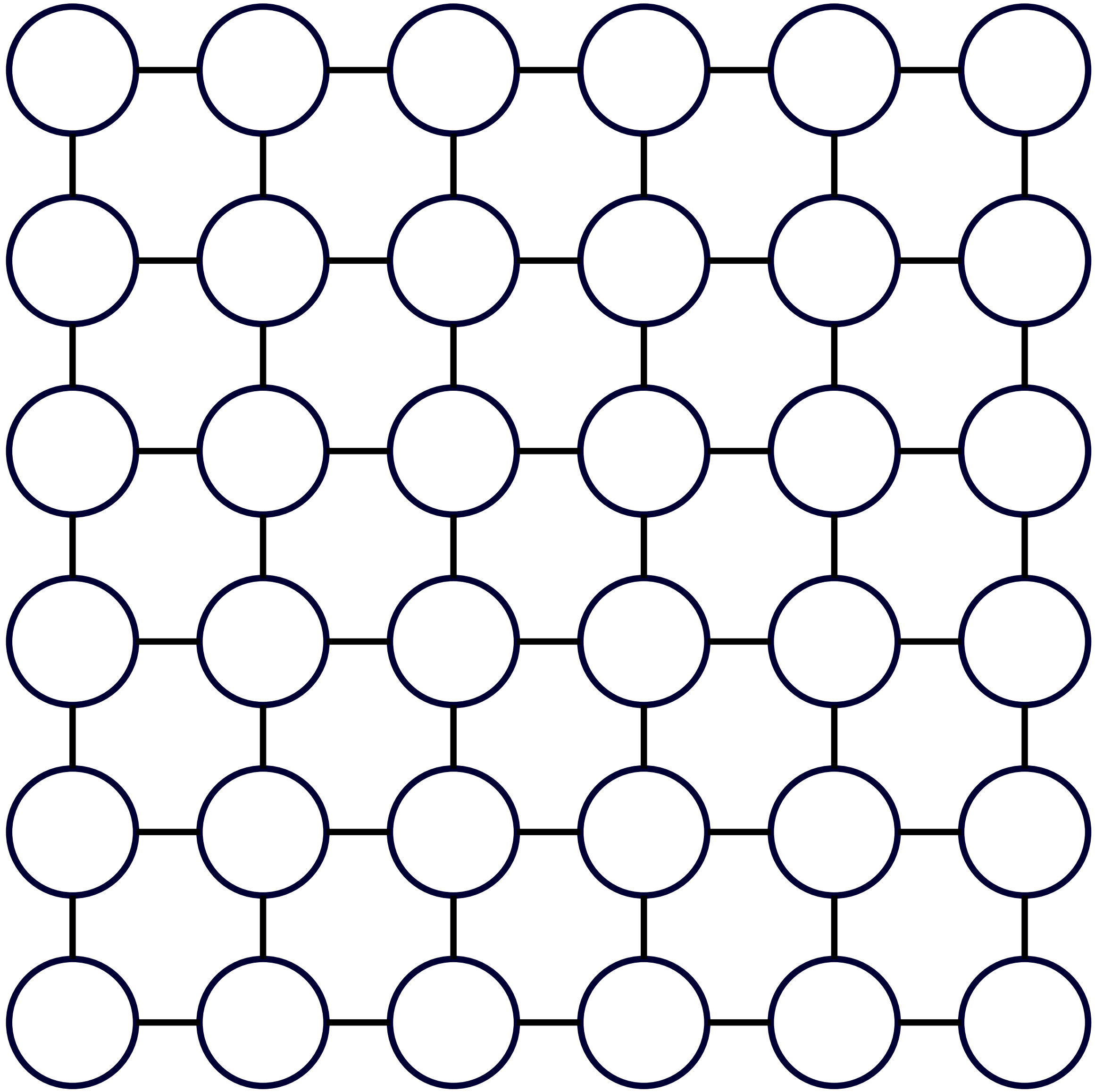
- Previous: Parallel Structured Duplicate Detection (Zhou and Hansen, 2007)
  - Used abstraction to divide labor.
  - Parallelized breadth-first search.

Overview

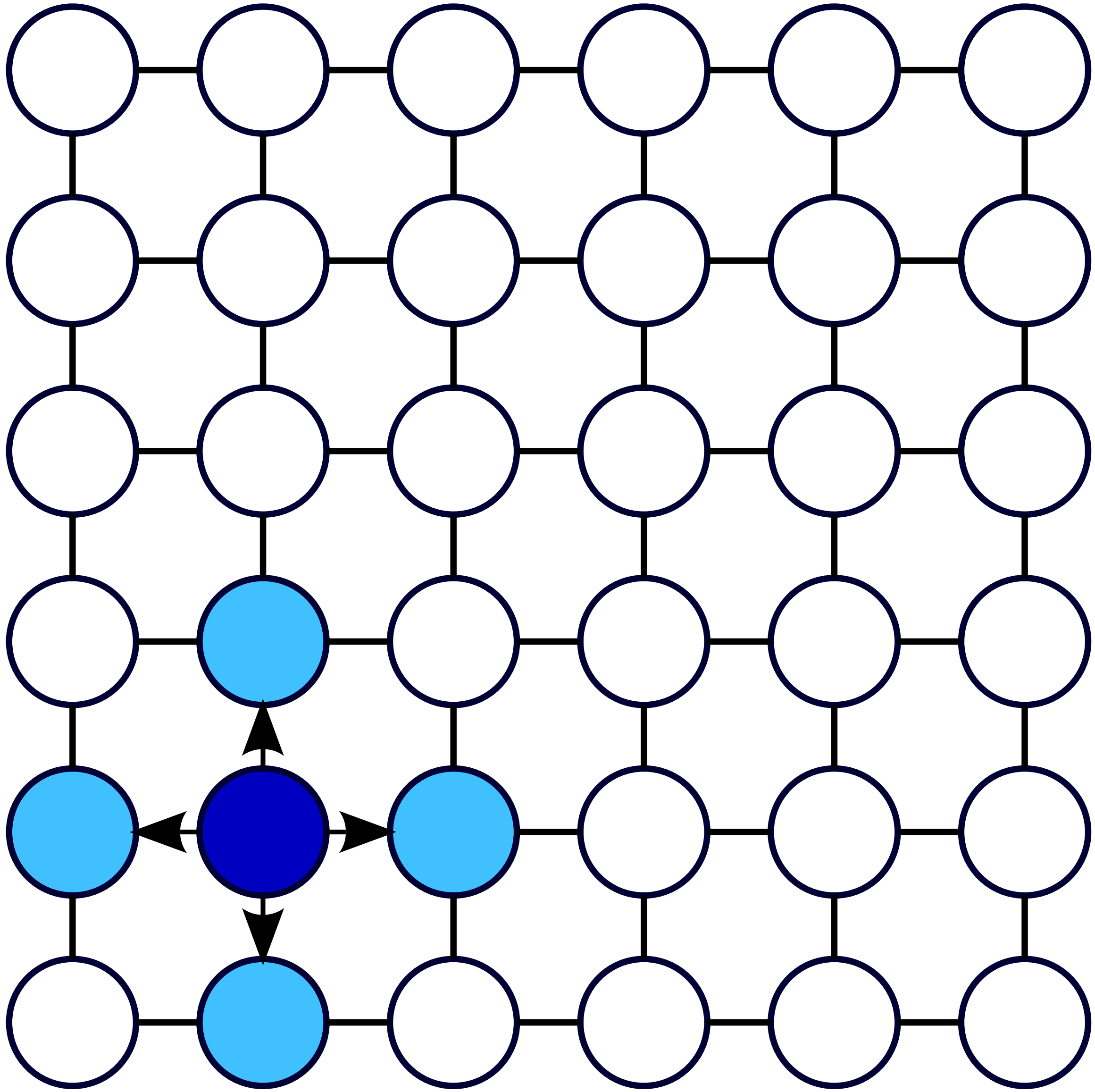
- Previous: Parallel Structured Duplicate Detection (Zhou and Hansen, 2007)
  - Used abstraction to divide labor.
  - Parallelized breadth-first search.
- New: Parallel Best NBlock First Search
  - Each thread tries to expand the best nodes.
  - Requires care to avoid livelock.

Previous: Parallel Structured Duplicate Detection

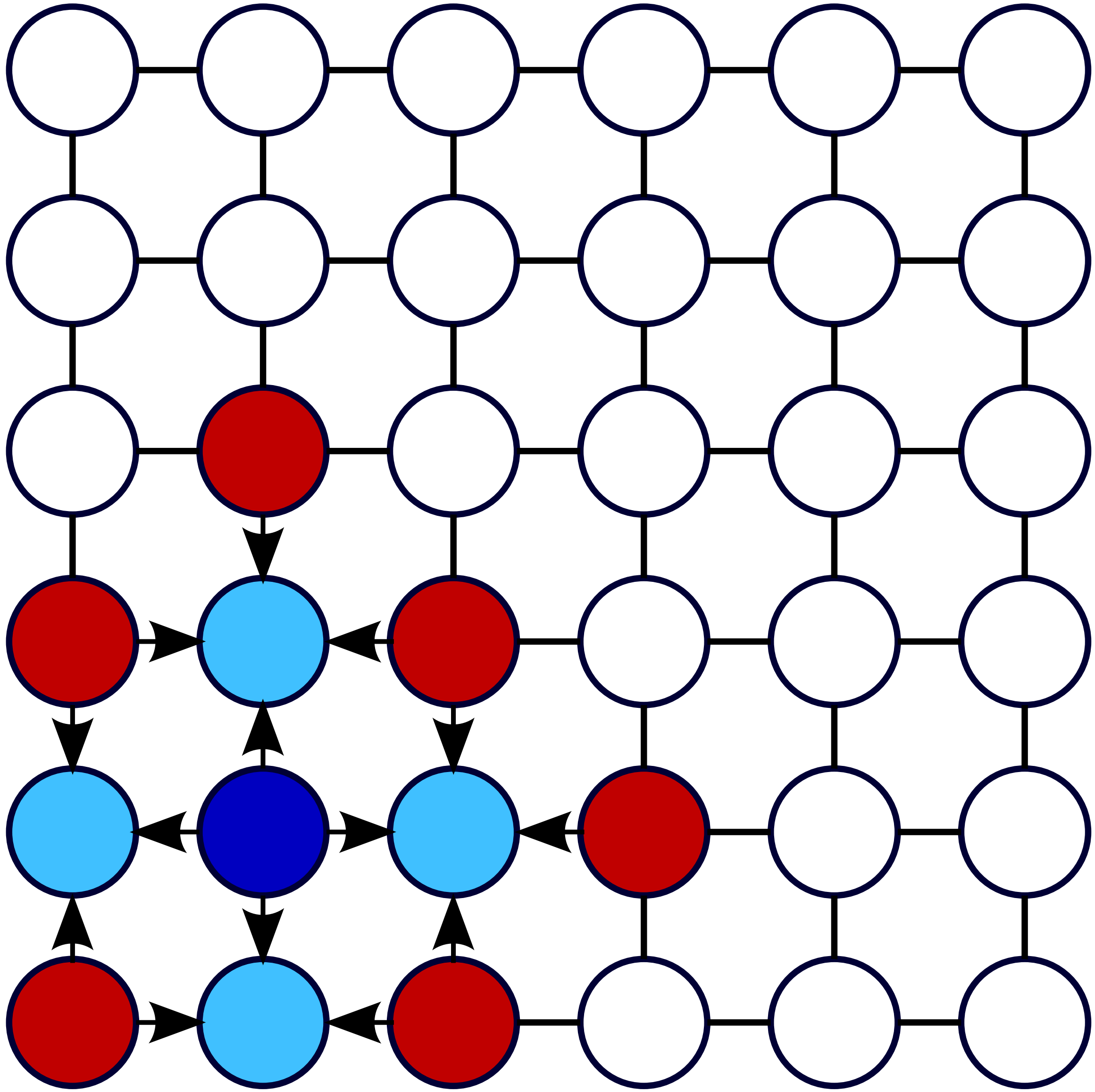
Naive Parallel Search



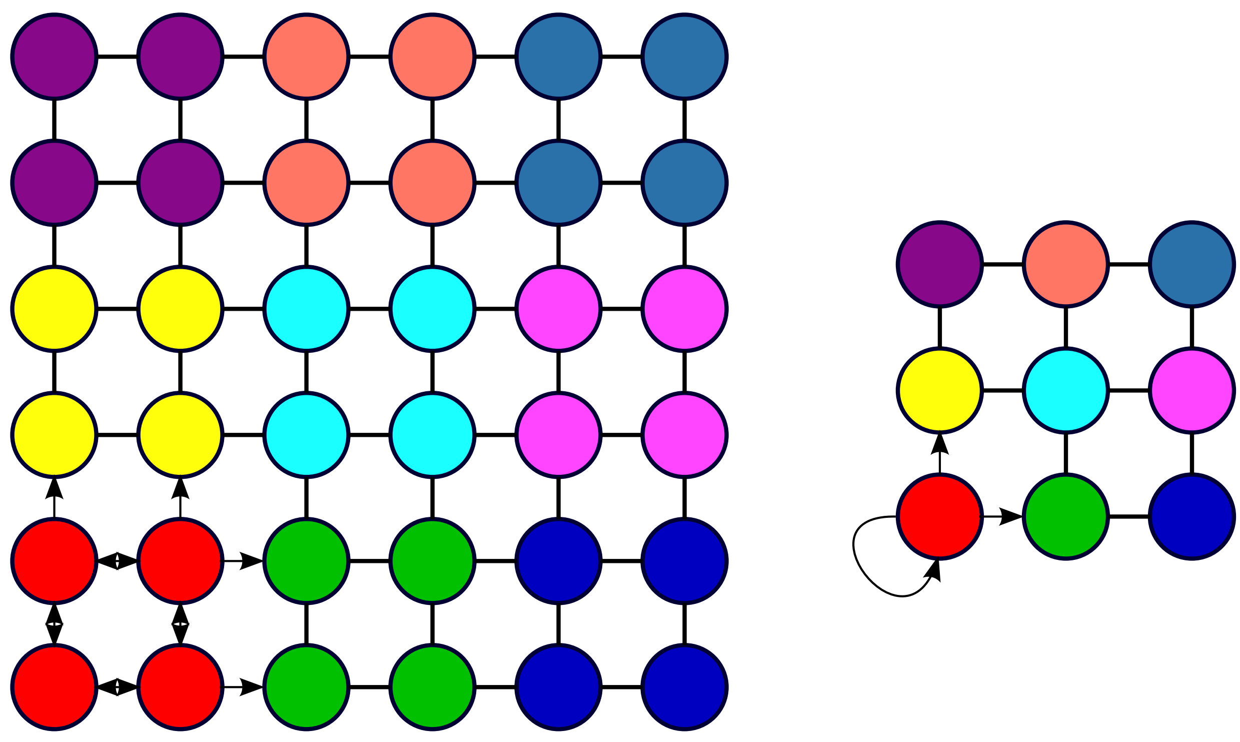
Naive Parallel Search



Naive Parallel Search

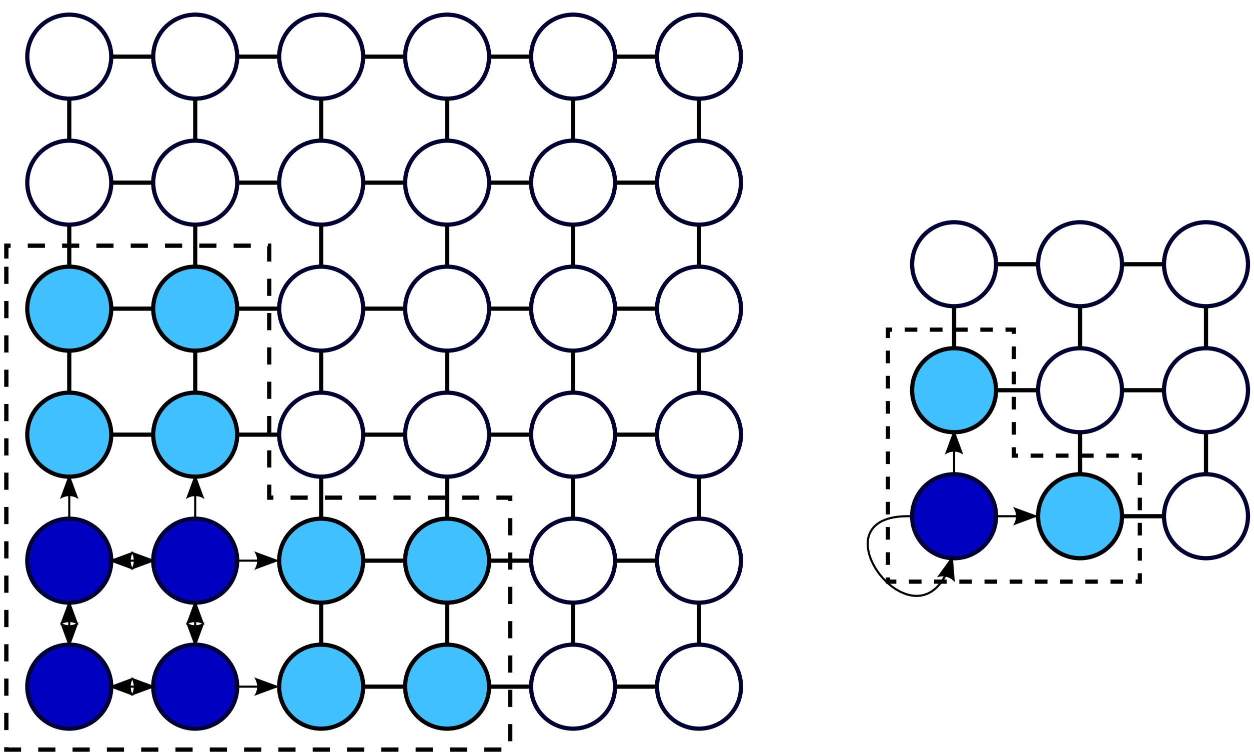


- Work is divided among threads using a special hash function based on abstraction.
  - Few possible destinations for children.



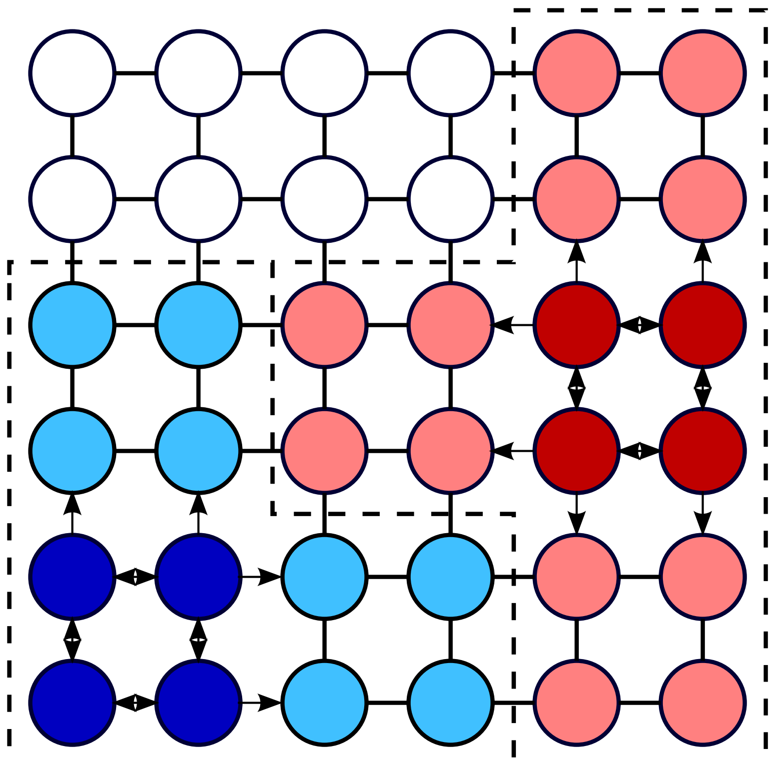
Duplicate Detection Scope

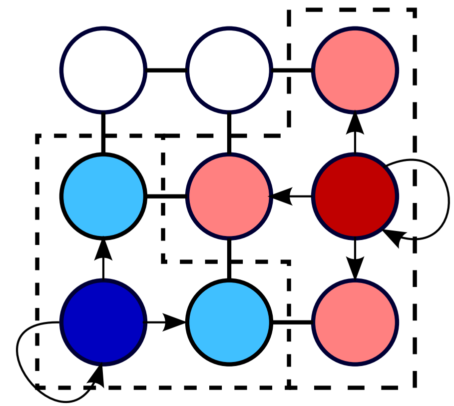
- Work is divided among threads using a special hash function based on abstraction.
  - Threads search groups of nodes called $n$blocks.



Disjoint Duplicate Detection Scopes

- Work is divided among threads using a special hash function based on abstraction.
  - Disjoint *duplicate detection scopes* searched in parallel.





Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.

Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.

Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.
- Only uses a single lock: when finding free disjoint scopes.

Parallel Structured Duplicate Detection

- Uses an abstract graph to decompose the search space.
- Threads proceed breadth-first in parallel.
  - All threads search the same depth layer.
  - All threads synchronize before moving to the next depth.
- Heuristic cost-to-go information is used for pruning.
  - Requires an upper-bound or iterative-deepening.
- Only uses a single lock: when finding free disjoint scopes.

We want a best-first ordering without layer-based synchronization and one lock.

New: Parallel Best NBlock First Search

Parallel Best NBlock First

1. Search disjoint $n$blocks in parallel.
   - Maintain a heap of free $n$blocks.
   - Greedily acquire best free $n$block (and its scope).

Parallel Best NBlock First

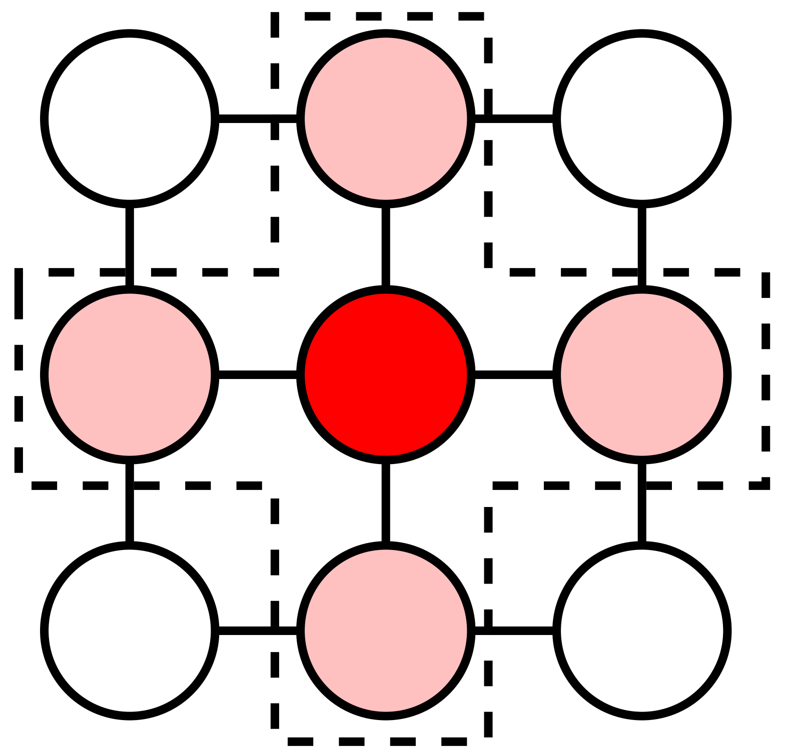
1. Search disjoint $n$blocks in parallel.
   - Maintain a heap of free $n$blocks.
   - Greedily acquire best free $n$block (and its scope).
2. Each $n$block is searched in $f(n)$ order.
   - Switch $n$blocks when a better one becomes free.
   - Perform a minimum amount of work before switching.
   - Approximates best-first order.

Parallel Best NBlock First

1. Search disjoint $n$blocks in parallel.
   - Maintain a heap of free $n$blocks.
   - Greedily acquire best free $n$block (and its scope).
2. Each $n$block is searched in $f(n)$ order.
   - Switch $n$blocks when a better one becomes free.
   - Perform a minimum amount of work before switching.
   - Approximates best-first order.
3. Stop when the incumbent solution is optimal.
   - Prune nodes on the cost of the incumbent
   - Incumbent is optimal when all nodes are pruned.

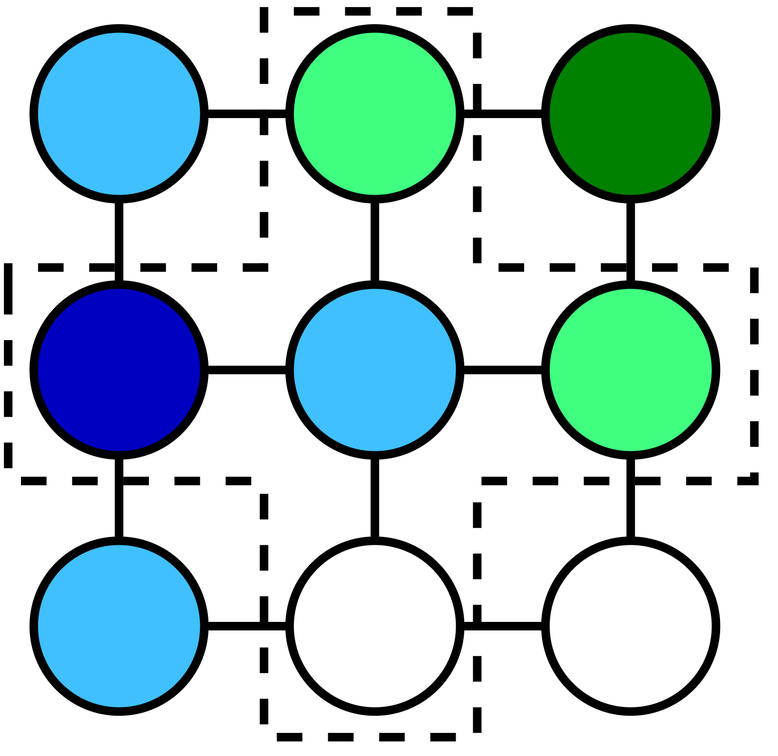
“Safe” PBNF

- Problem:



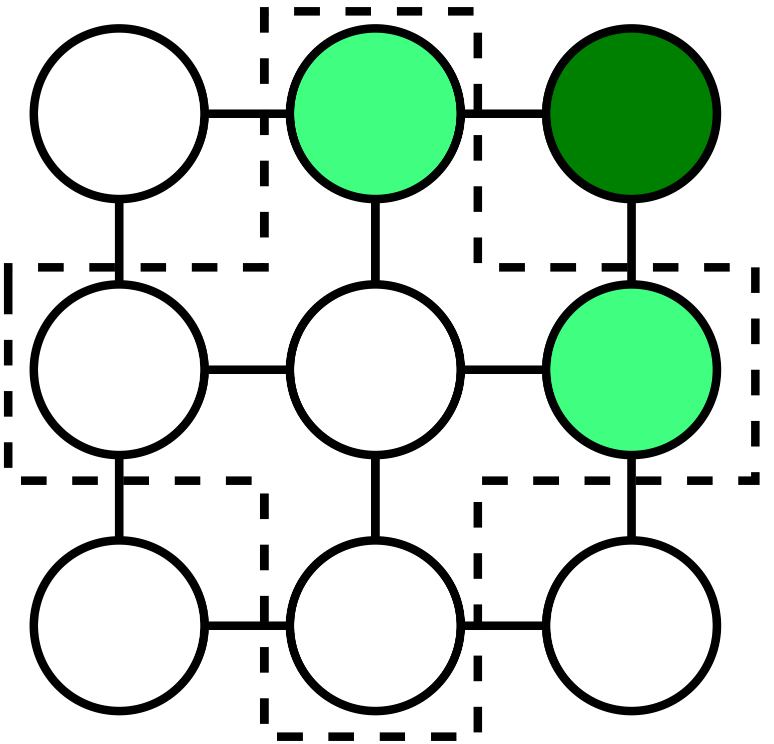
"Safe" PBNF

- Problem:



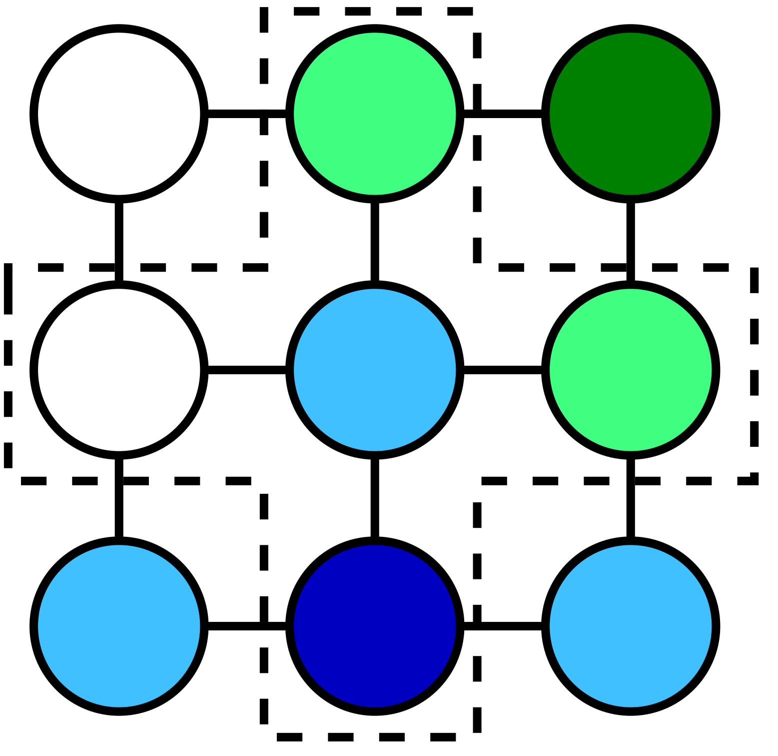
“Safe” PBNF

- Problem:



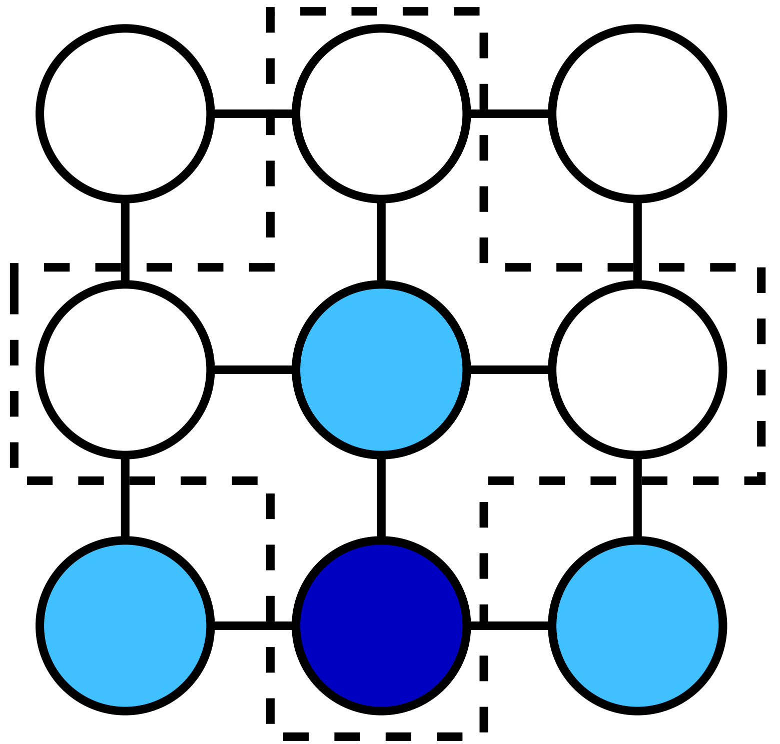
"Safe" PBNF

- Problem:

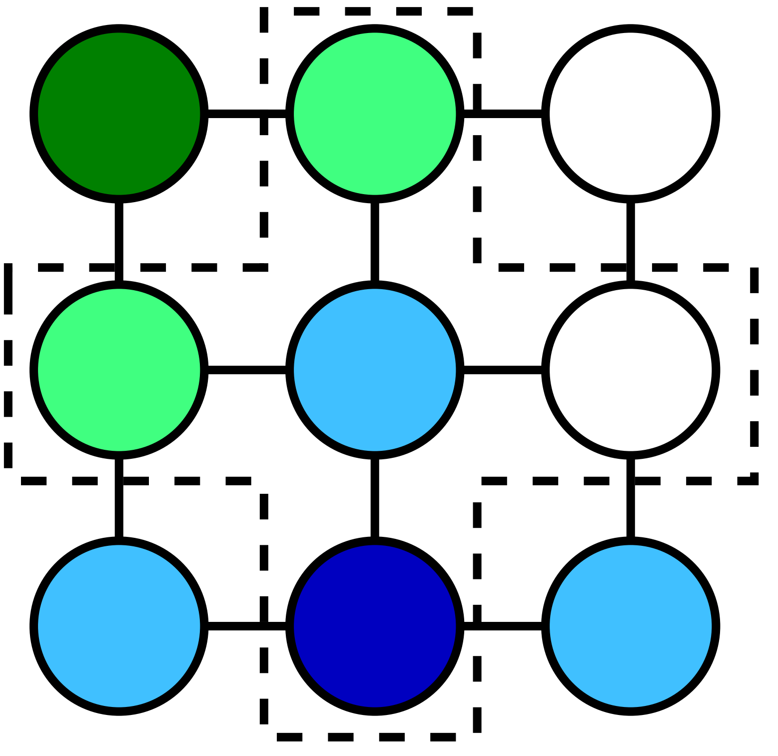


“Safe” PBNF

- Problem:

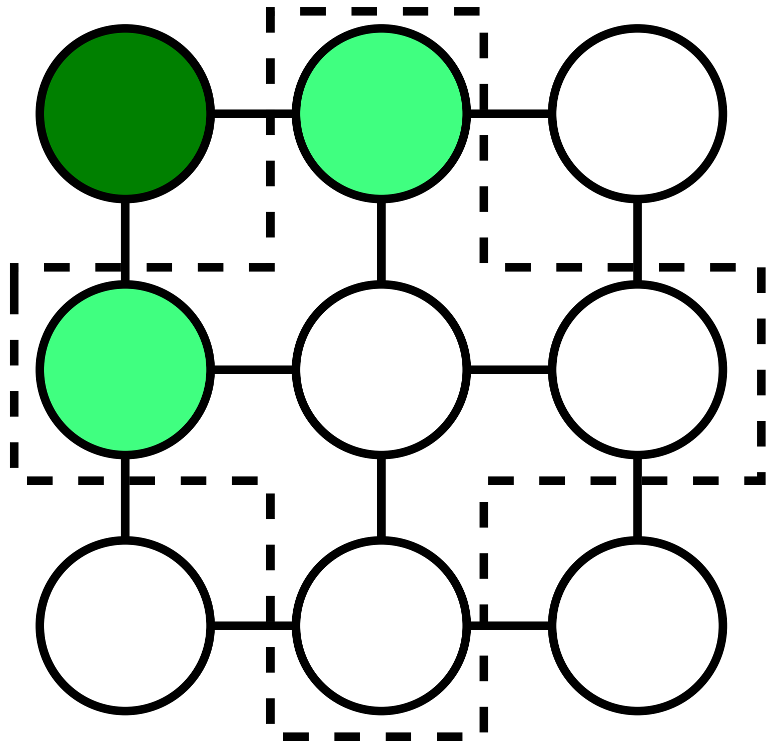


- Problem:



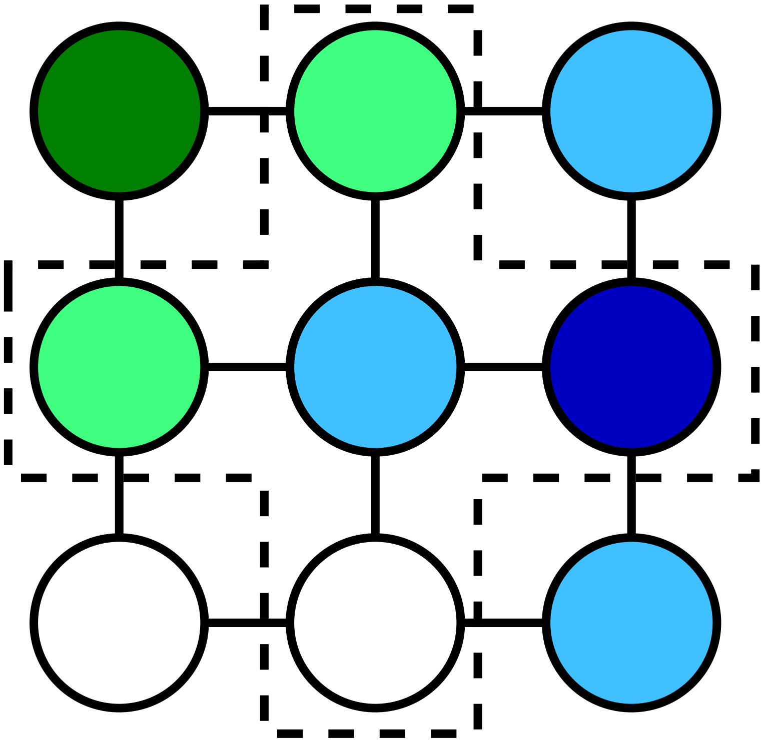
"Safe" PBNF

- Problem:



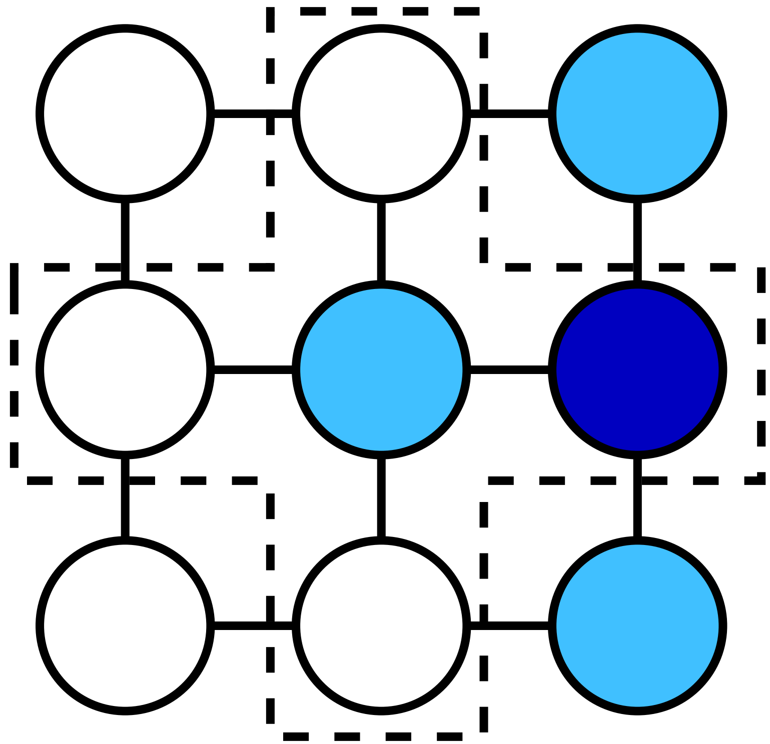
"Safe" PBNF

- Problem:



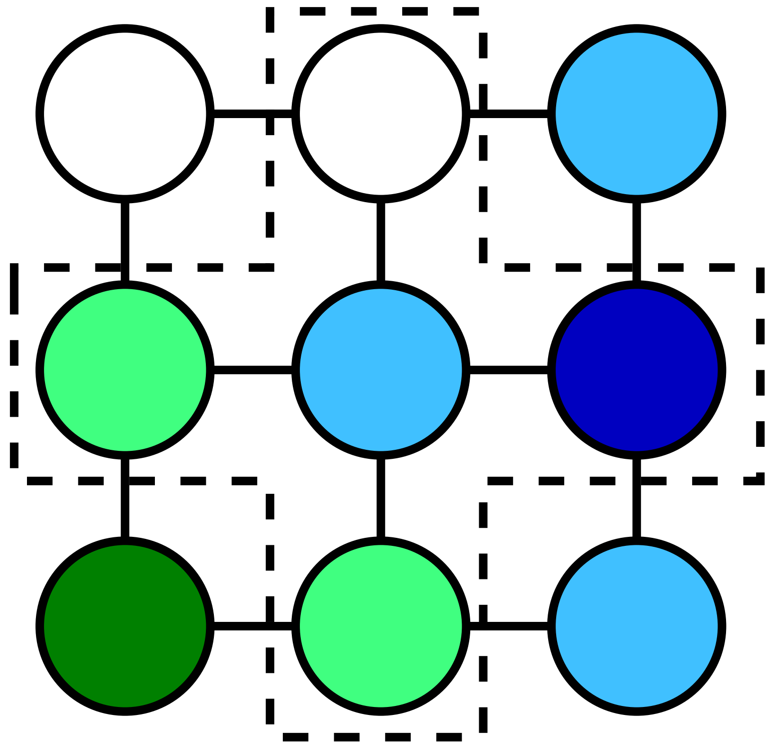
"Safe" PBNF

- Problem:



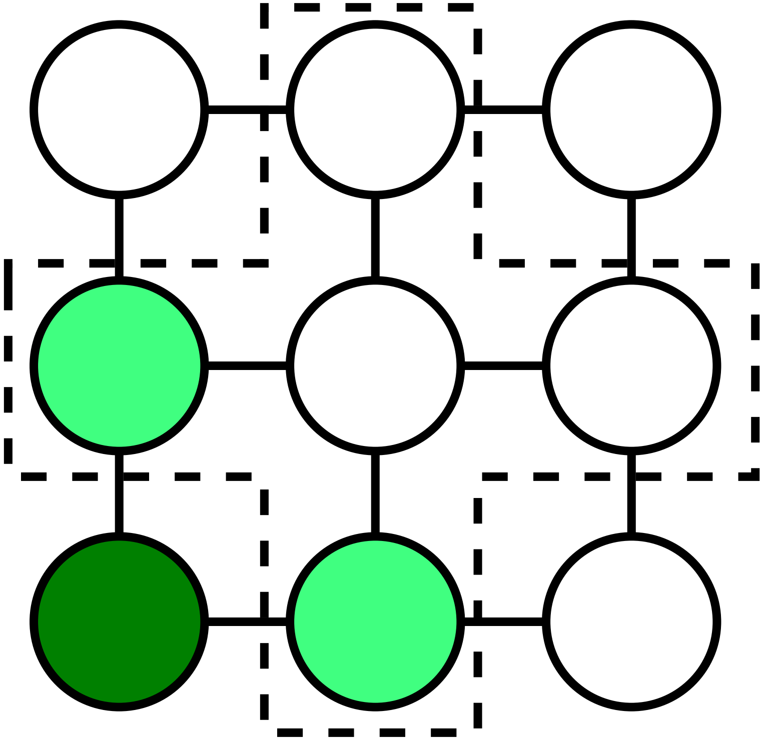
"Safe" PBNF

- Problem:



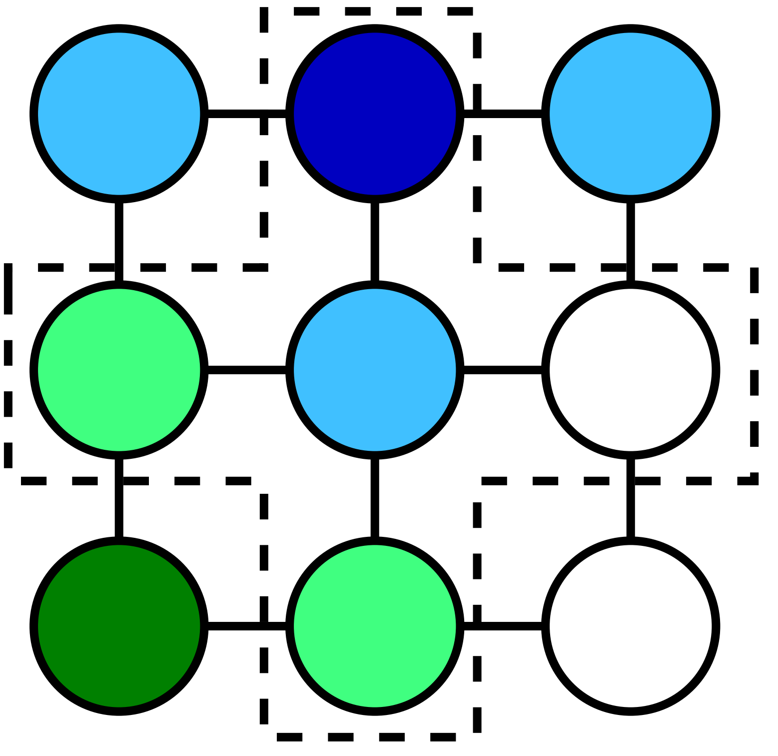
"Safe" PBNF

- Problem:



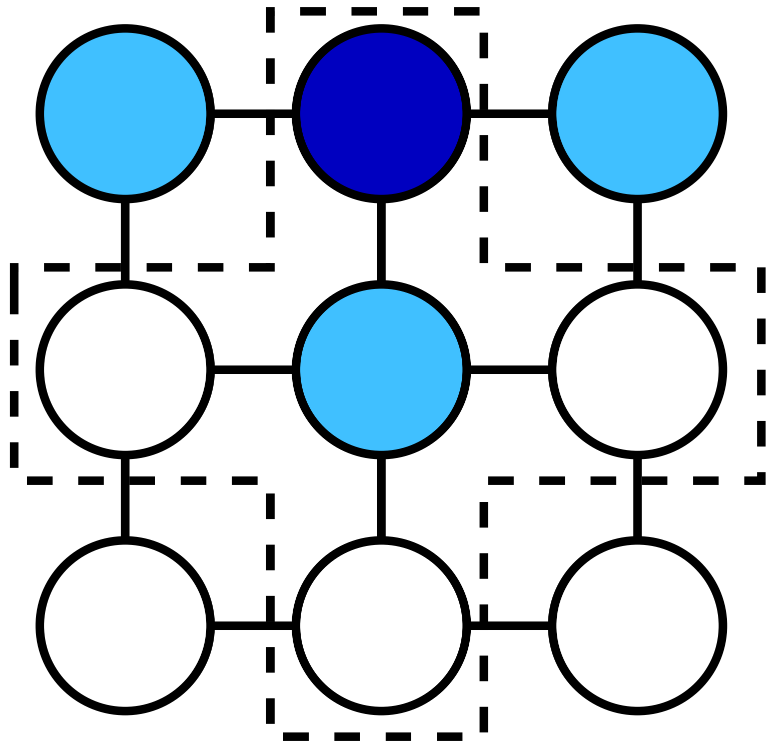
"Safe" PBNF

- Problem:



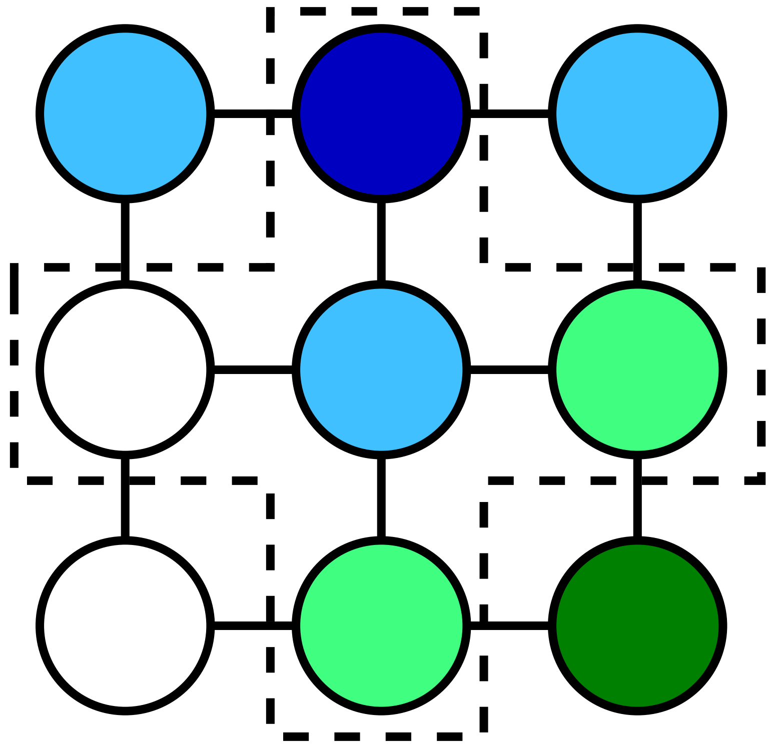
"Safe" PBNF

- Problem:



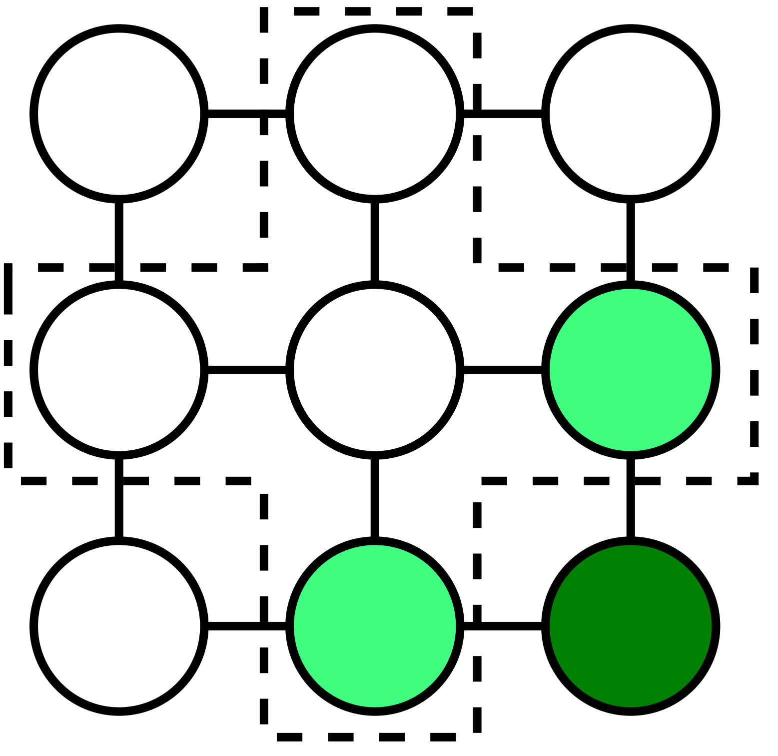
“Safe” PBNF

- Problem:



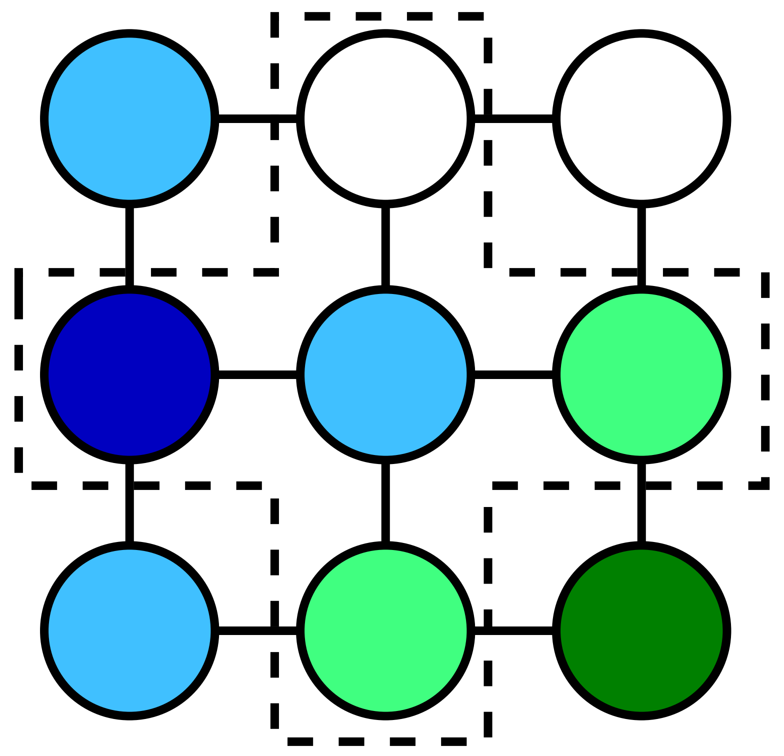
“Safe” PBNF

- Problem:



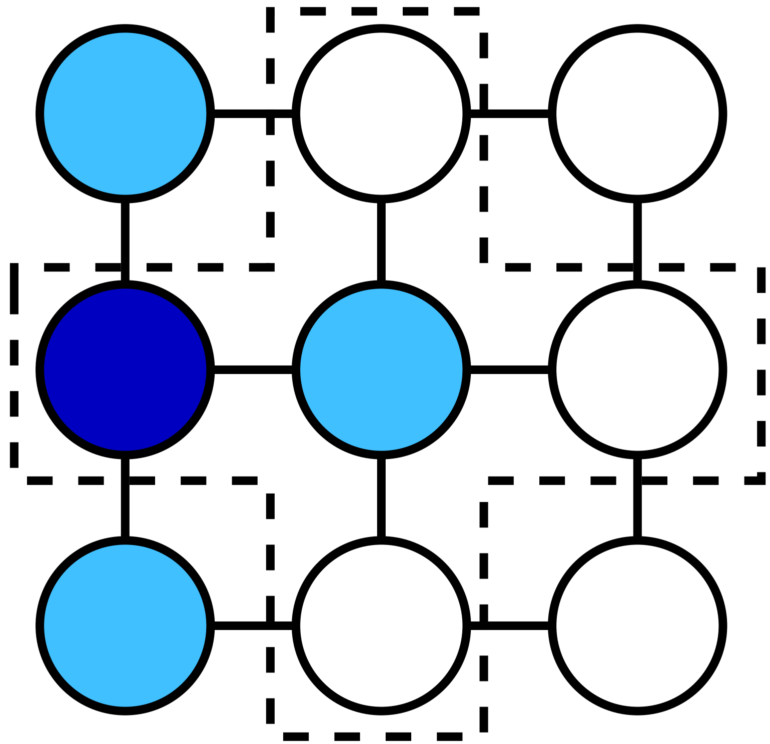
“Safe” PBNF

- Problem:



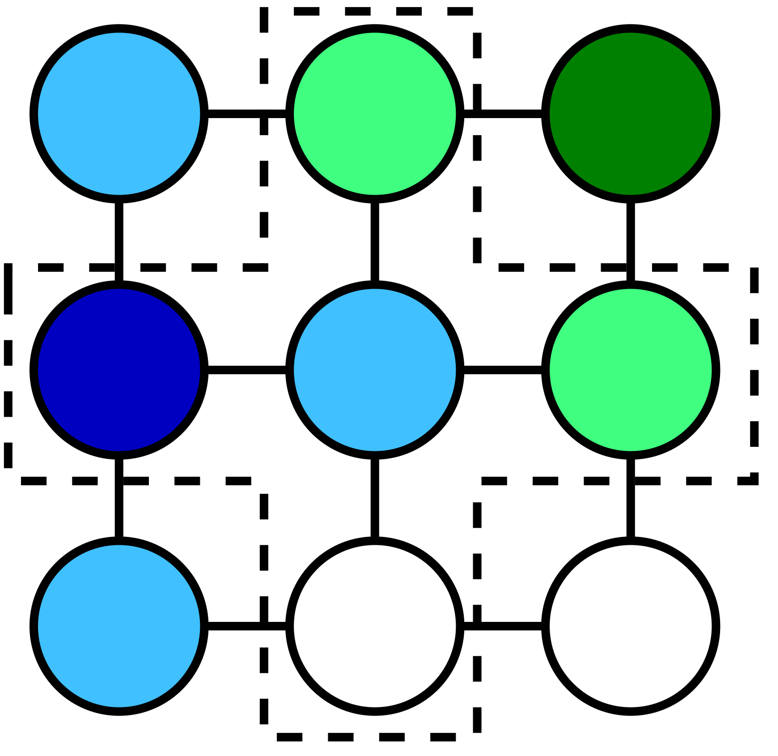
“Safe” PBNF

- Problem:



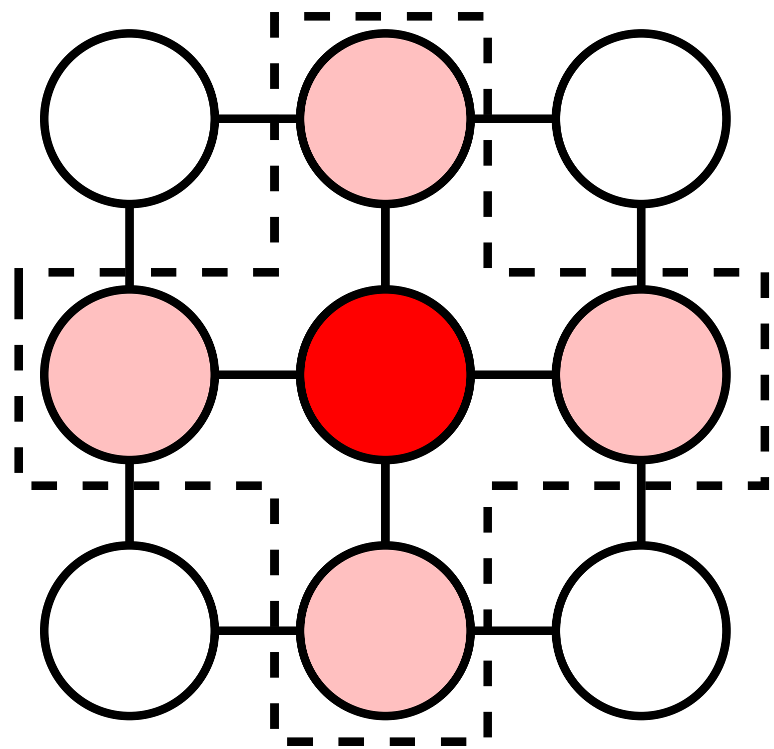
“Safe” PBNF

- Problem:



“Safe” PBNF

- Problem:

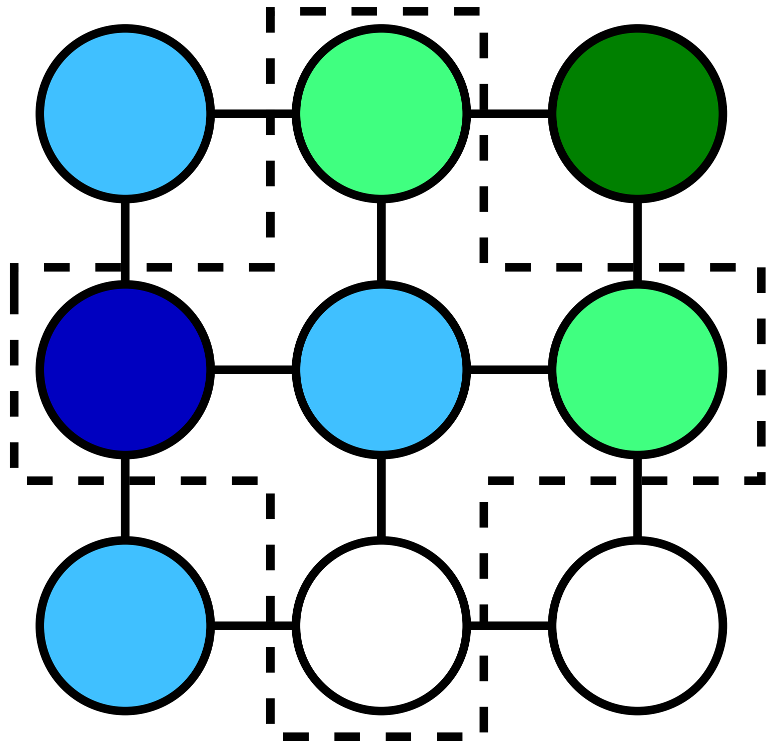


“Safe” PBNF

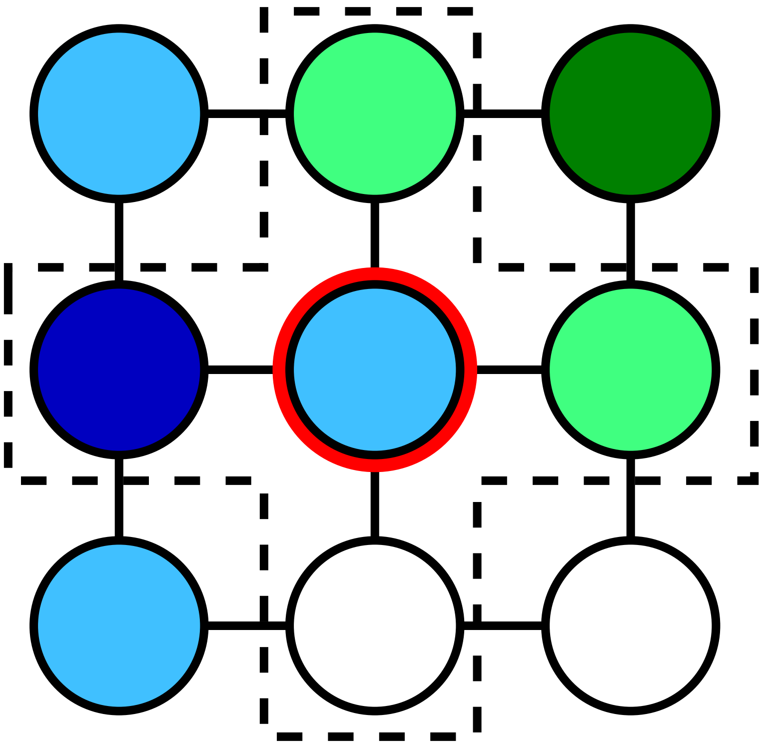
- No guarantee that a given $n$block will become free.
  - In infinite search spaces, there can be livelock.
- Solution: check for *hot* $n$blocks
  - Flag better $n$blocks as *hot*
  - Release an $n$block to free an interfered hot $n$block.

“Safe” PBNF

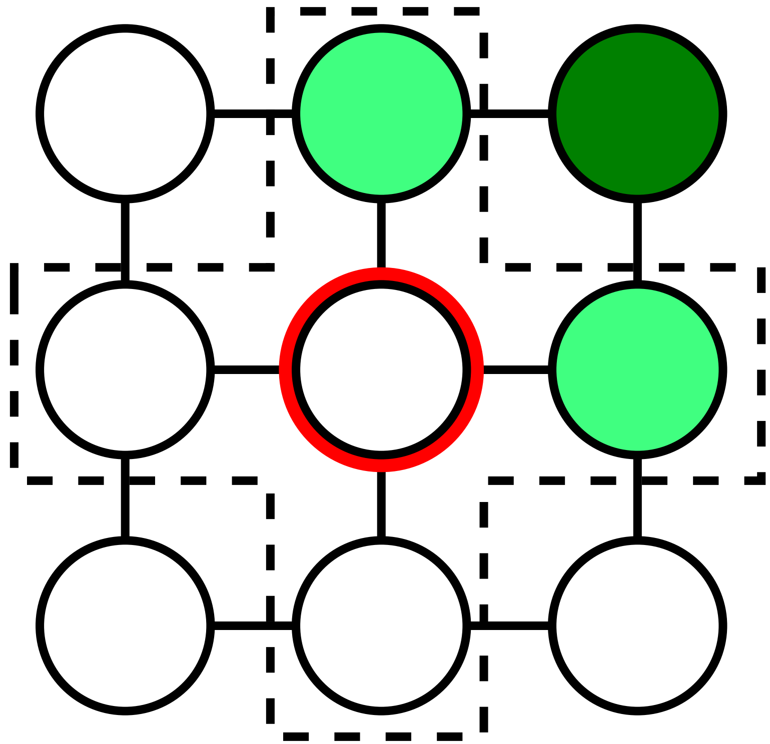
- Solution:



- Solution:

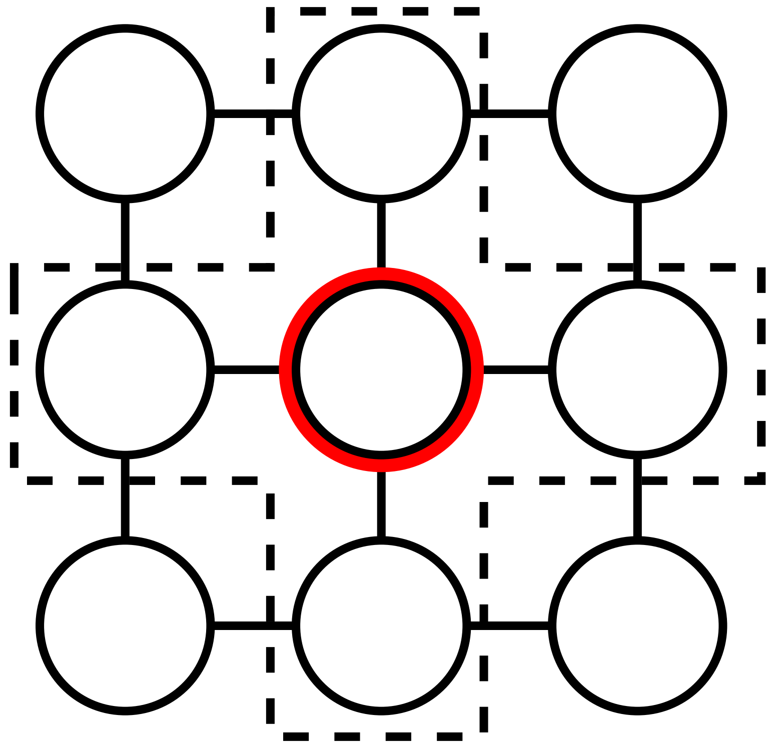


- Solution:



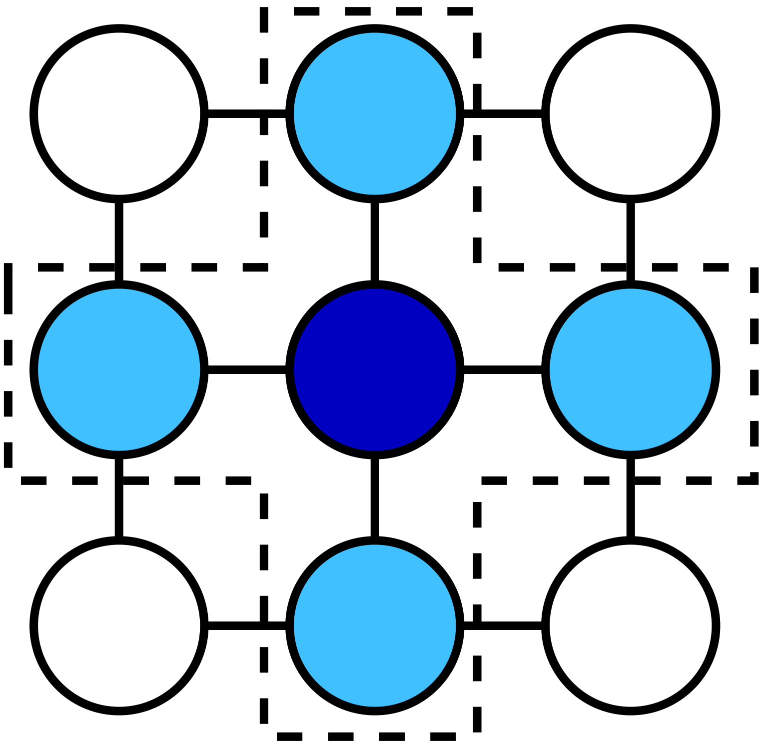
“Safe” PBNF

- Solution:



“Safe” PBNF

- Solution:



Empirical Evaluation

Empirical Evaluation

Software

- C++
- POSIX threads
- jemalloc (Grids and Tiles) / custom allocator (STRIPS planning)
- Fedora 9

Hardware

- Dual quad-core Intel Xeon E5320 1.86GHz 64-bits
- 16Gb RAM

Domains

- Grid pathfinding
  - Abstraction: coarser grid
- 15-puzzles (easy 43 of Korf's 100)
  - Abstraction: ignore some tile numbers
- STRIPS planning
  - Abstraction: generated automatically

Previous Algorithms

PA*

- Basic A* with a lock on open and closed lists.

Lock-free PA*

- PA* with lock-free data structures.

KBFS (Felner et al., 2003)

- Expand the K best open nodes in parallel.

PRA* (Evett et al., 1995)

- Hash nodes to distribute among processors.
- Synchronized message queues for "incoming" nodes.

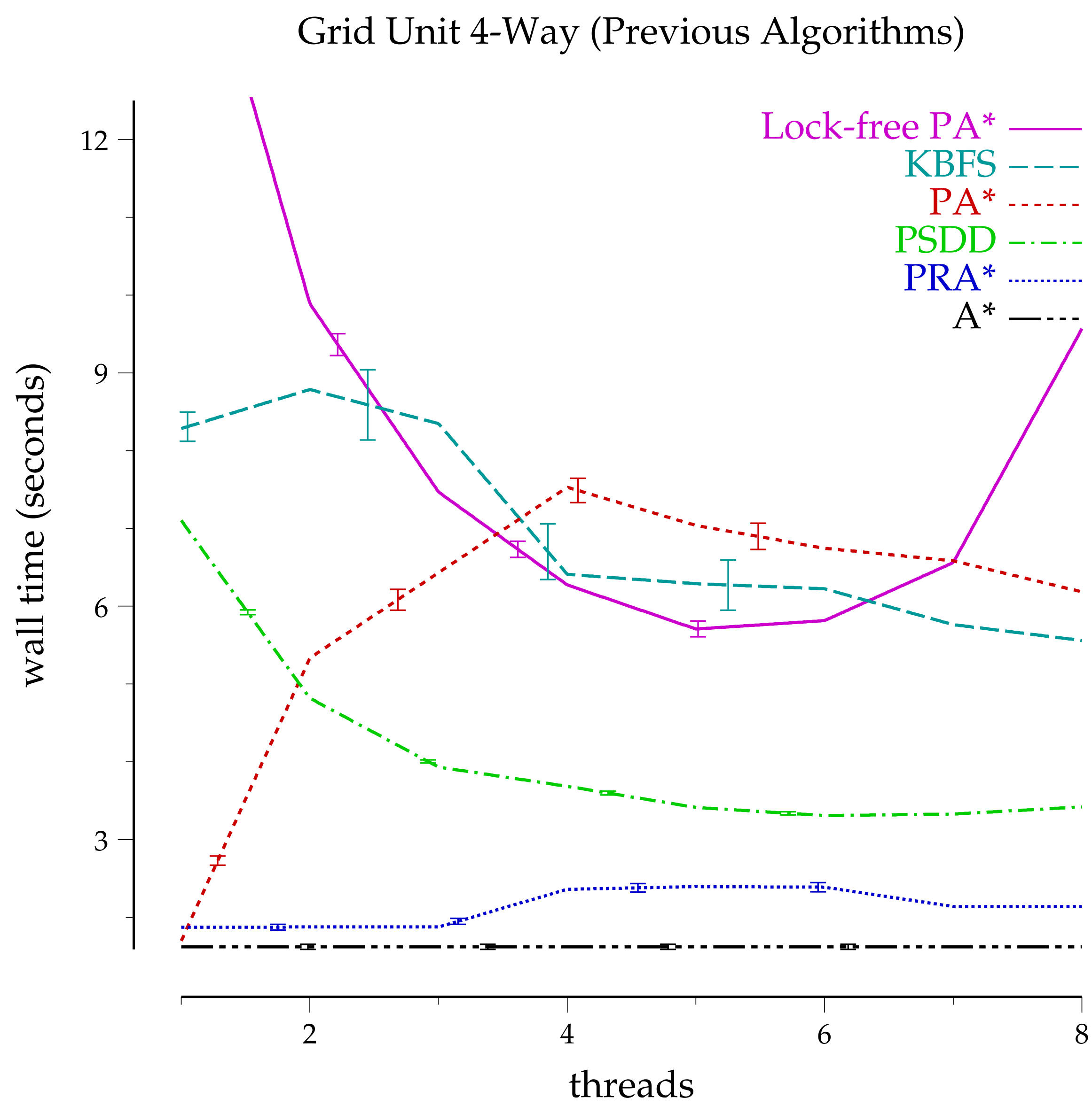
PSDD (Zhou and Hansen, 2007)

- Abstraction to find disjoint portions of a search space.
- Breadth-first search
- All threads synchronize at each layer

IDPSDD

- PSDD with iterative-deepening for bounds.

Four-way Grid Pathfinding (Previous Algorithms)



APRA*

- PRA* with a novel abstraction based hashing.
- Limits contention for message queues.

BFPSDD

- PSDD with $f(n)$ layers instead of depth layers.

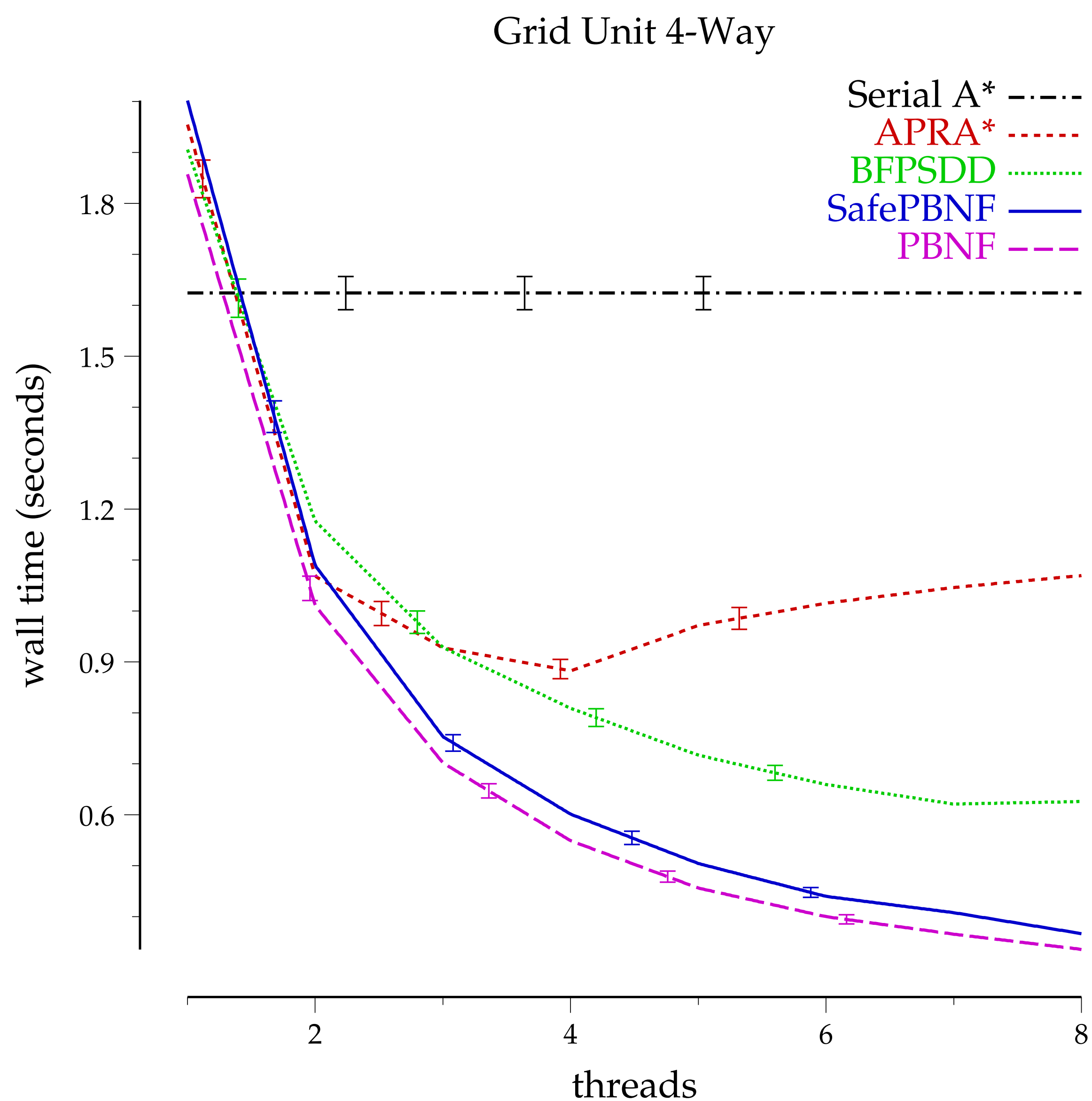
PBNF

- Acquire the best free $n$block.

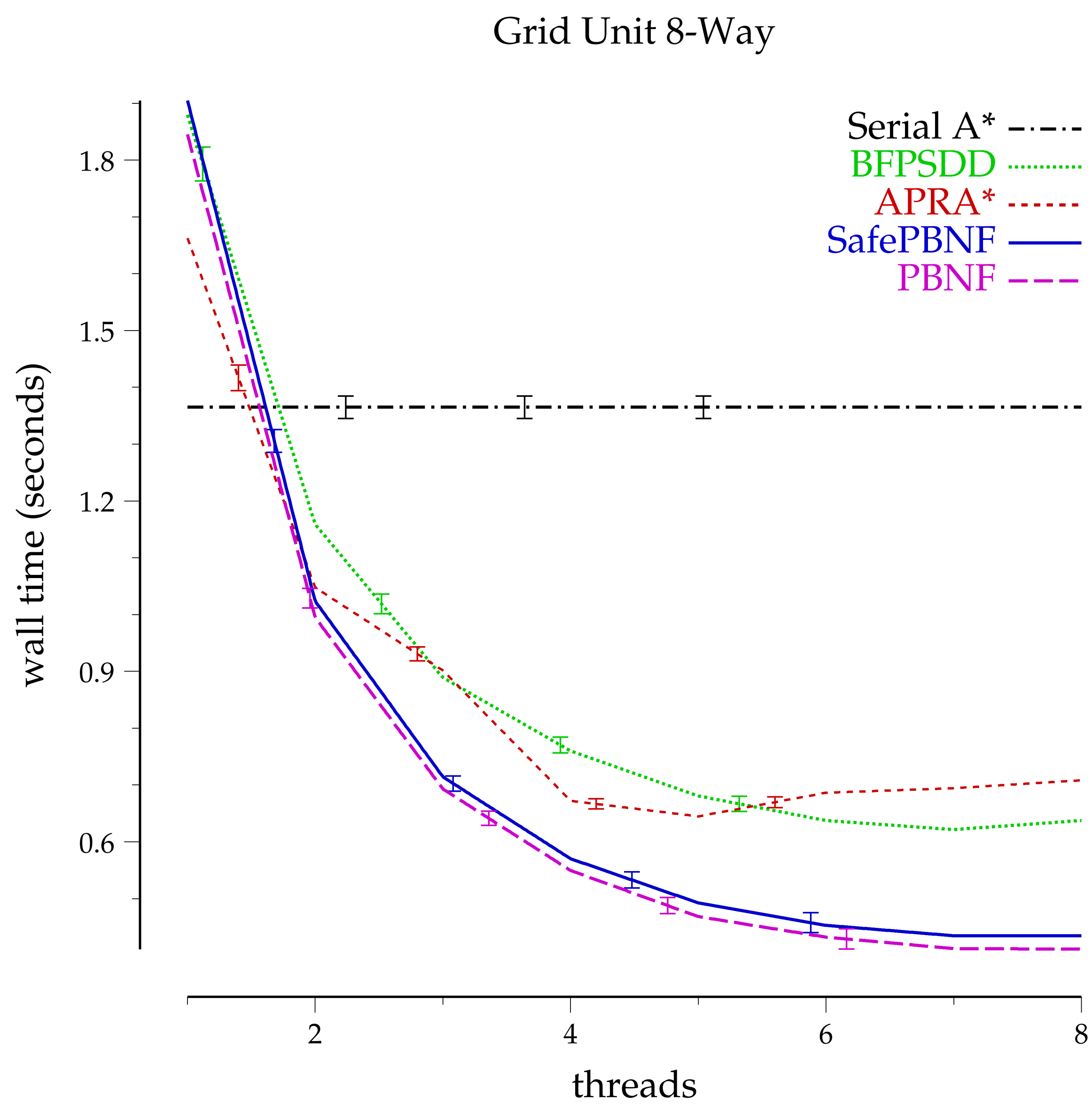
Safe PBNF

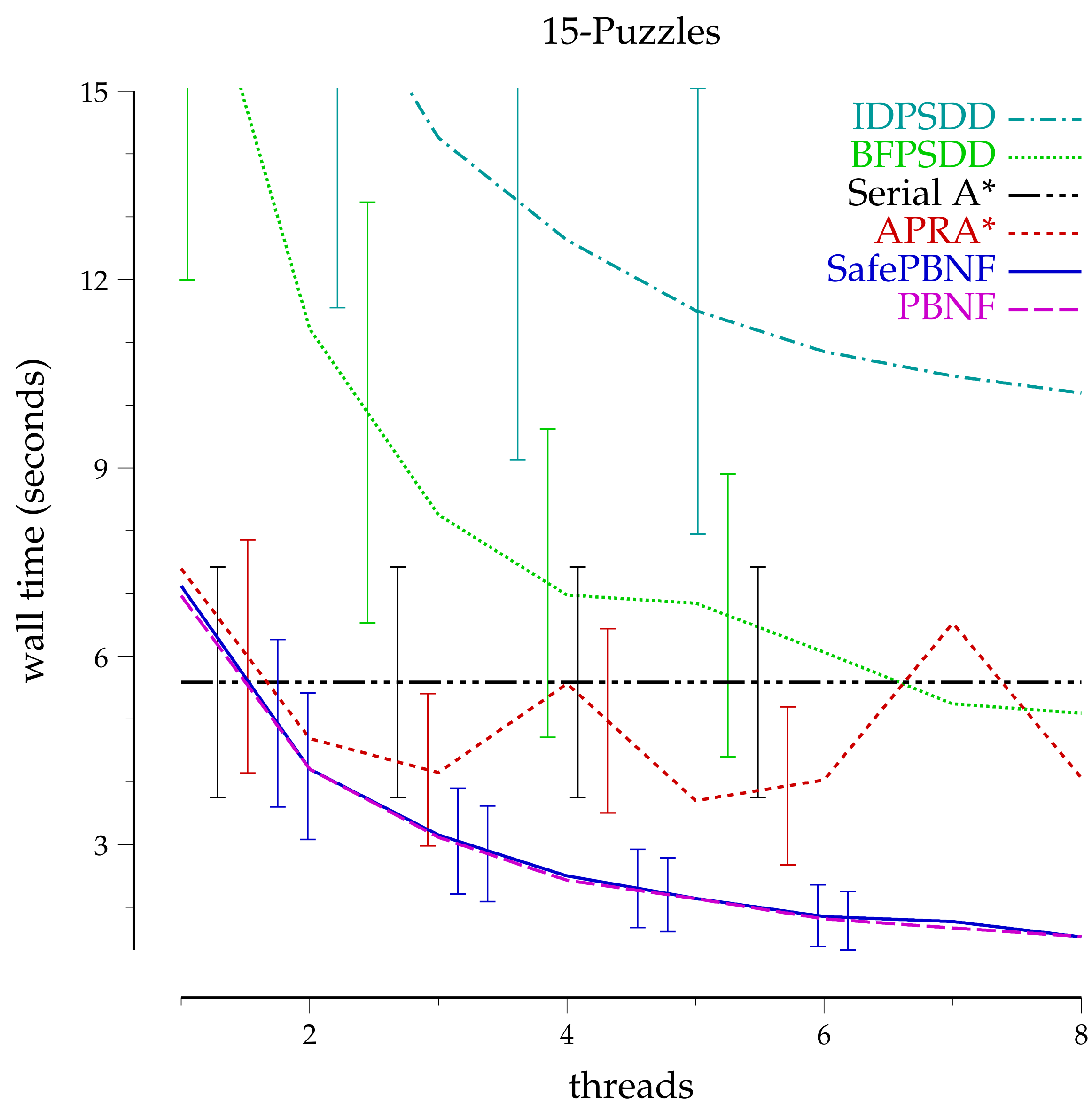
- PBNF with livelock prevention.

Four-way Grid Pathfinding (New Algorithms)



Eight-way Grid Pathfinding




15-Puzzles
IDPSDD
BFPSDD
Serial A*
APRA*
SafePBNF
PBNF
wall time (seconds)
15
12
9
6
3
threads
2
4
6
8

STRIPS Planning

	threads	logistics-6	blocks-14	gripper-7	satellite-6	elevator-12	freecell-3	depots-7	driverlog-11	gripper-8
A*	1	2.3	5.2	118	131	336	199	M	M	M
APRA*	1	1.5	7.1	60	96	213	150	301	322	528
	3	0.76	5.5	51	49	269	112	144	103	M
	5	1.2	3.8	41	66	241	61	M	M	M
	7	0.84	3.7	28	49	169	40	M	M	M
PNBF	1	1.3	6.3	40	68	157	186	M	M	230
	3	0.72	3.8	16	34	56	64	M	M	96
	5	0.58	2.7	11	21	35	44	M	M	61
	7	**0.53**	2.6	**8.6**	17	**27**	**36**	M	M	**48**
SafePBNF	1	1.2	6.2	40	77	150	127	156	154	235
	3	0.64	2.7	17	24	54	47	63	60	98
	5	0.56	2.2	11	17	34	38	43	39	64
	7	0.62	**2.0**	9.2	**14**	**27**	37	**35**	**31**	52
BFPSDD	1	2.1	7.8	42	62	152	131	167	152	243
	3	1.1	4.3	18	24	59	57	67	62	101
	5	0.79	3.9	12	20	41	48	48	43	71
	7	0.71	3.4	10	**14**	32	45	43	35	59

Wall time in seconds

Conclusion

Coming Attractions

- Ethan Burns, Seth Lemons, Wheeler Ruml and Rong Zhou, *Suboptimal and Anytime Heuristic Search on Multi-Core Machines*, ICAPS 2009
  - Proof of correctness.
  - Bounded suboptimal PBNF.
  - Anytime PBNF.

Coming Attractions

- Ethan Burns, Seth Lemons, Wheeler Ruml and Rong Zhou, *Suboptimal and Anytime Heuristic Search on Multi-Core Machines*, ICAPS 2009
  - Proof of correctness.
  - Bounded suboptimal PBNF.
  - Anytime PBNF.

Future Direction

- External memory PBNF.

New: Parallel Best NBlock First.

New: Parallel Best NBlock First.

- Fast
  - Beats all other algorithms used for comparison.

Conclusion

New: Parallel Best NBlock First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.

Conclusion

New: Parallel Best NBlock First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.
- Easy to use
  - Only requires a user-provided abstraction.

Conclusion

New: Parallel Best NBlock First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.
- Easy to use
  - Only requires a user-provided abstraction.
- "Hot $n$blocks"
  - Prevents livelock.
  - Not much overhead.

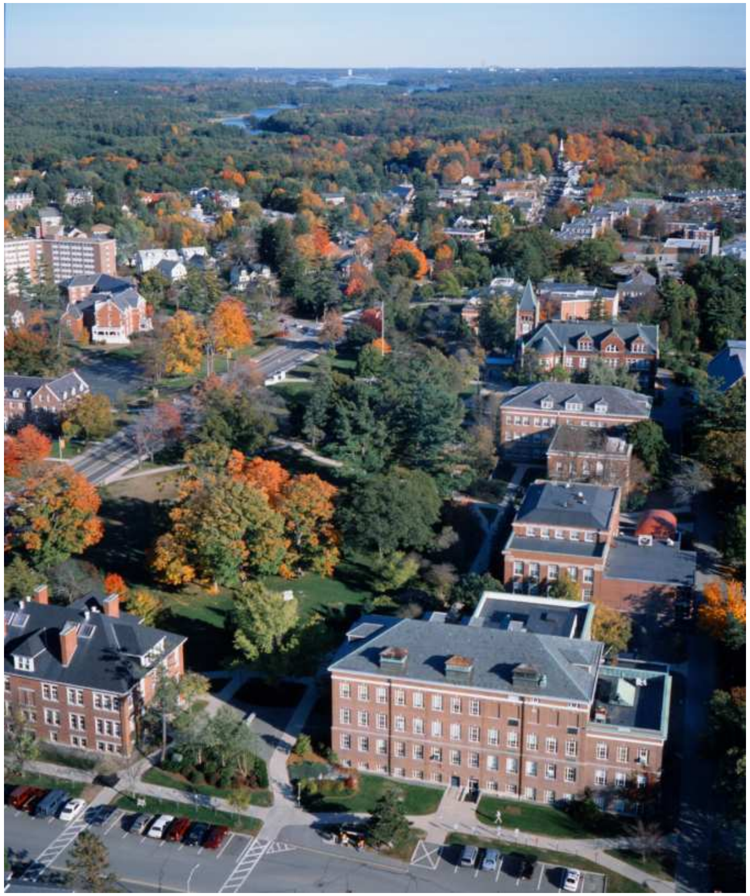
Conclusion

New: Parallel Best NBlock First.

- Fast
  - Beats all other algorithms used for comparison.
- Scales well
  - Tested out to eight threads.
- Easy to use
  - Only requires a user-provided abstraction.
- "Hot $n$blocks"
  - Prevents livelock.
  - Not much overhead.
- Source is freely available: `http://www.cs.unh.edu/~eaburns`

The University of New Hampshire

Tell your students to apply to grad school in CS at UNH!



- friendly faculty
- funding
- individual attention
- beautiful campus
- low cost of living
- easy access to Boston, White Mountains
- strong in AI, infoviz, networking, systems