Solving Atomix with Pattern Databases

Alex Gliesch, Marcus Ritt

Institute of Informatics
Universidade Federal do Rio Grande do Sul

October 7, 2016

What is Atomix?

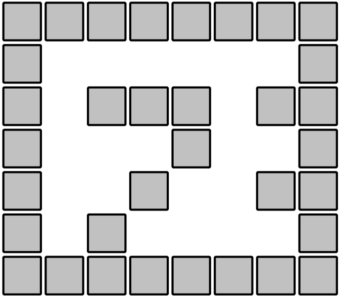
- A puzzle taking place on a rectangular 2D grid

What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)

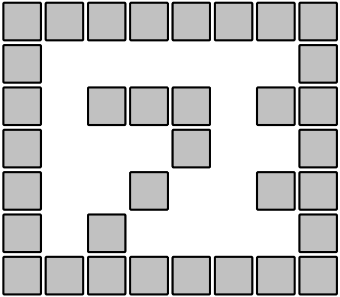
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)



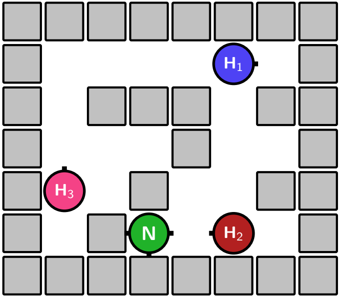
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms



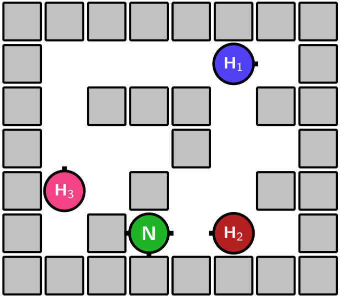
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms



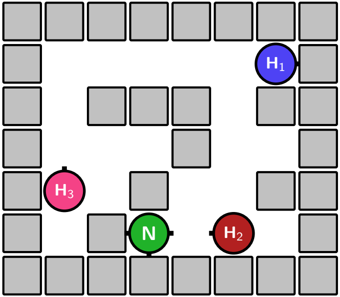
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



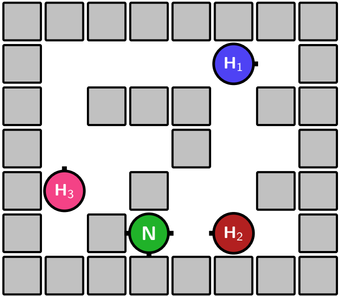
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



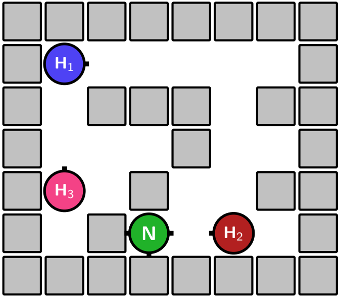
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



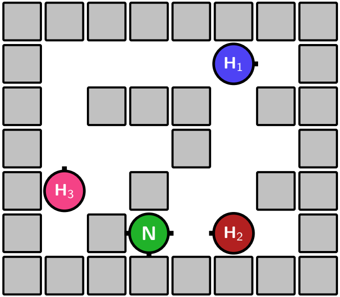
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



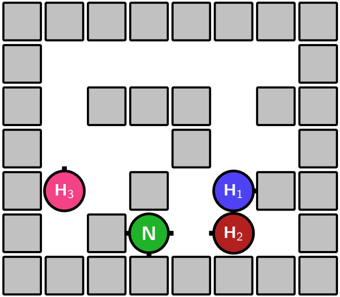
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



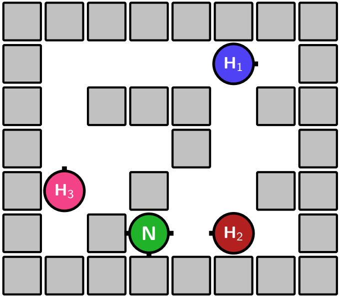
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



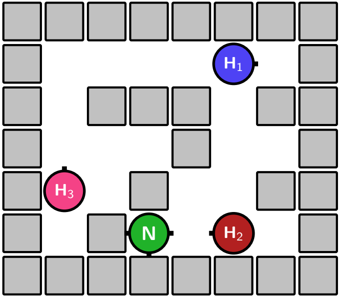
What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations



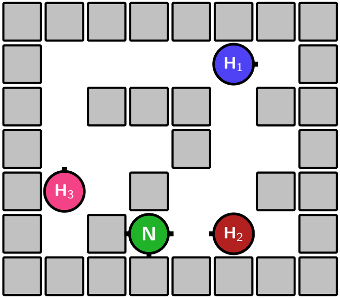
What is Atomix?

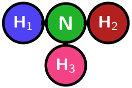
- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations
- The objective is to assemble the atoms in a specific way, which is called a molecule



What is Atomix?

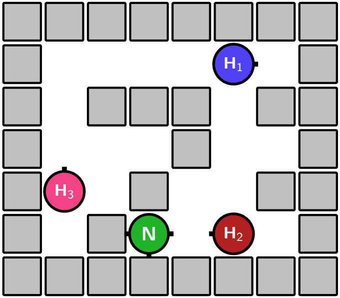
- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations
- The objective is to assemble the atoms in a specific way, which is called a molecule

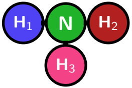




What is Atomix?

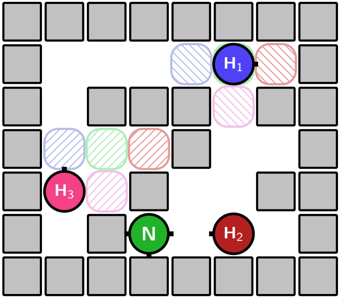
- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations
- The objective is to assemble the atoms in a specific way, which is called a molecule
- The molecule (usually) can be formed in more than one place

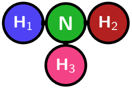




What is Atomix?

- A puzzle taking place on a rectangular 2D grid
- On the grid, there are obstacles (walls)
- As well as pieces called atoms
- The player may move those atoms using sliding operations
- The objective is to assemble the atoms in a specific way, which is called a molecule
- The molecule (usually) can be formed in more than one place





This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)
- Improve existing heuristics using a method called pattern databases (PDBs)

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A*, IDA*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:
  - An abstraction of Atomix in order to build PDBs

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:
  - An abstraction of Atomix in order to build PDBs
  - 2 types of PDBs: static and dynamic

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:
  - An abstraction of Atomix in order to build PDBs
  - 2 types of PDBs: static and dynamic
  - Different method of handling multiple goal states

This work

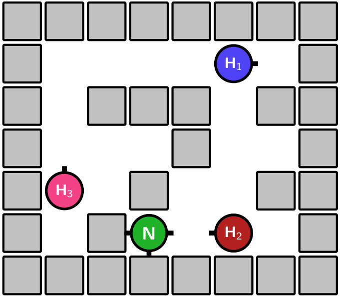
- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A^*, IDA^*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:
  - An abstraction of Atomix in order to build PDBs
  - 2 types of PDBs: static and dynamic
  - Different method of handling multiple goal states
  - Use of Partial Expansion A^* in order to reduce memory consumption

This work

- Motivation: find the minimum number of slides needed to form the molecule, using state space search (A*, IDA*)
- Improve existing heuristics using a method called pattern databases (PDBs)
- In this work, we propose:
  - An abstraction of Atomix in order to build PDBs
  - 2 types of PDBs: static and dynamic
  - Different method of handling multiple goal states
  - Use of Partial Expansion A* in order to reduce memory consumption
- This presentation: focus on PDBs, other contributions with more details in the paper

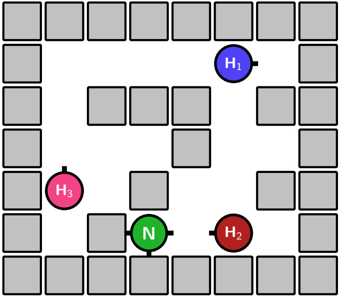
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**



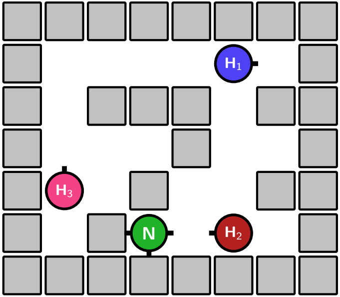
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



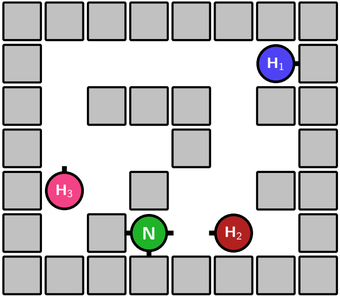
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



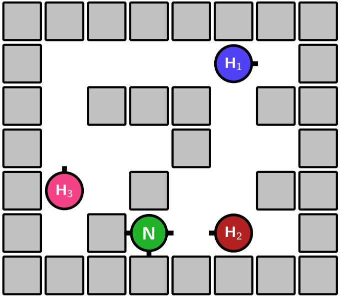
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



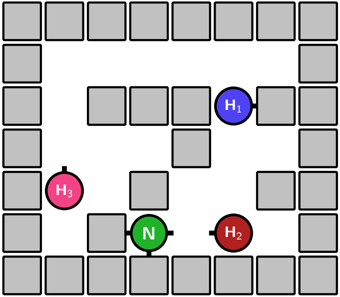
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



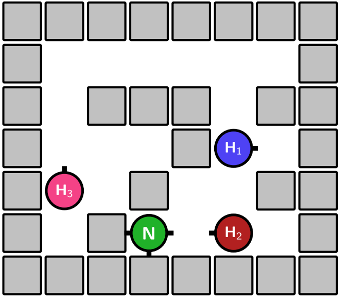
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



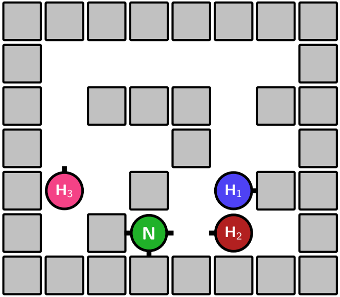
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



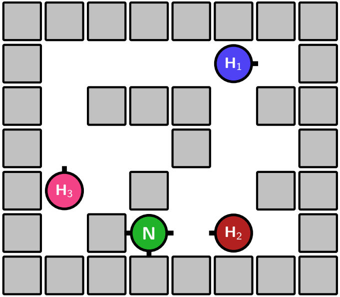
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



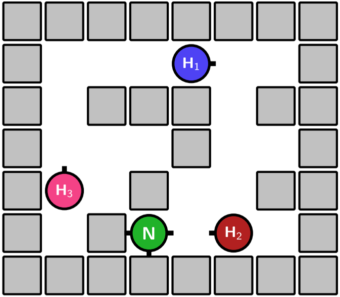
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



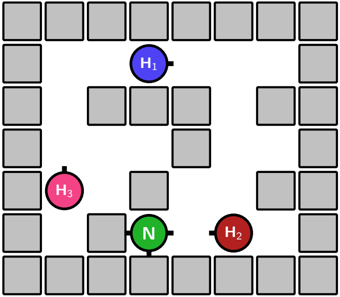
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



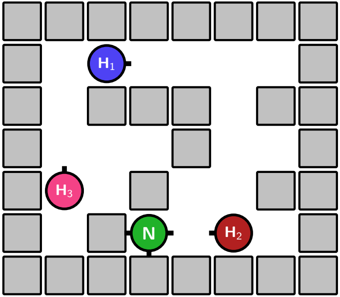
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



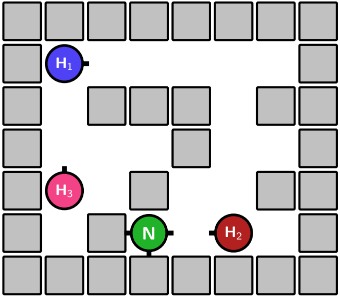
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



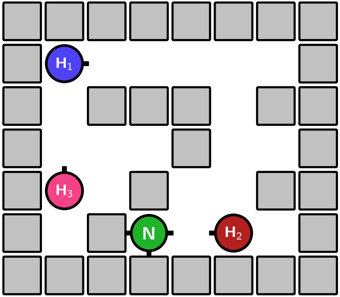
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way



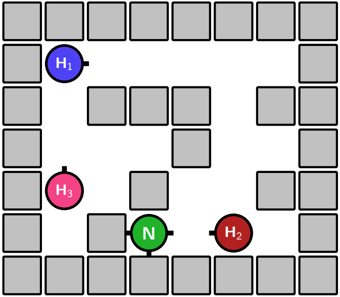
Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way
- Generalized distance: minimum number of relaxed moves from one position to another

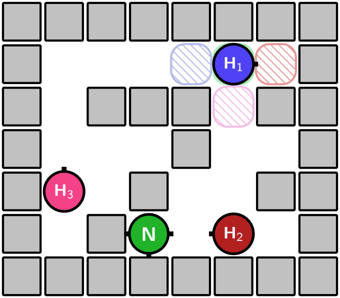


Previous work on Atomix

- Hüffner et al. (2001) studied state space search on Atomix and proposed the **generalized moves heuristic**
- Relaxation: slides allow atoms to stop anywhere on the way
- Generalized distance: minimum number of relaxed moves from one position to another
- **Heuristic value of a state:** sum of generalized distances from every atom, independently, to its final position in a molecule

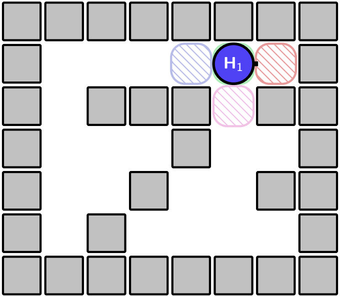


Generalized moves: example



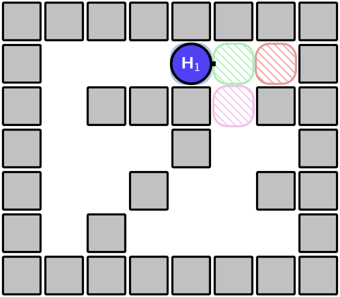
Each atom is handled independently.

Generalized moves: example



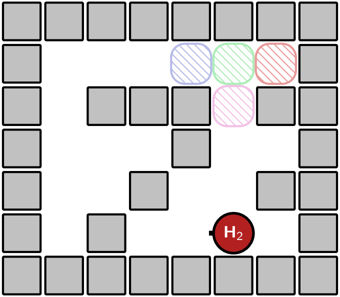
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$

Generalized moves: example



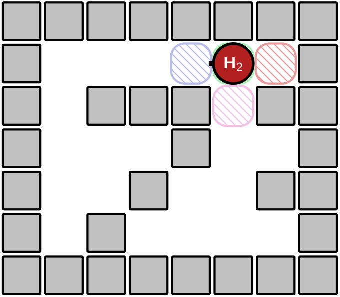
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$

Generalized moves: example



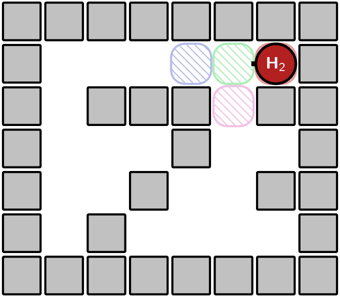
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$

Generalized moves: example



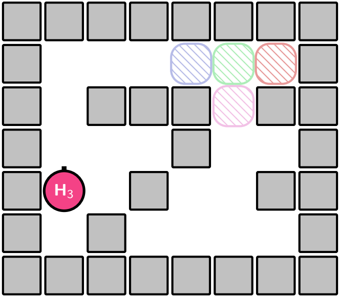
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$

Generalized moves: example



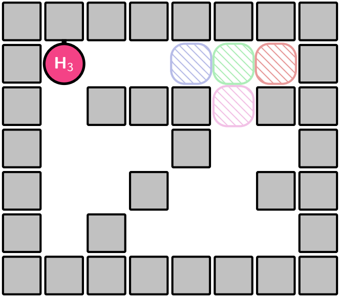
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$

Generalized moves: example



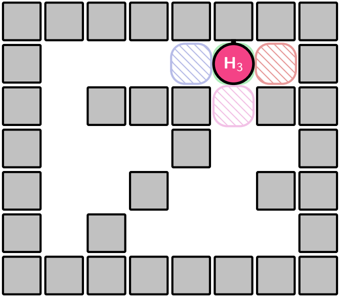
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H_1}$, 2 for $\mathbf{H_2}$, 3 for $\mathbf{H_3}$

Generalized moves: example



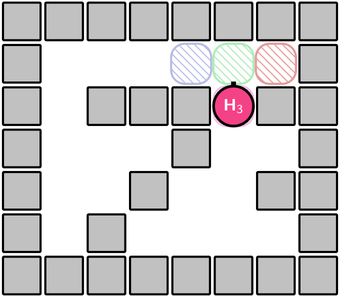
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$

Generalized moves: example



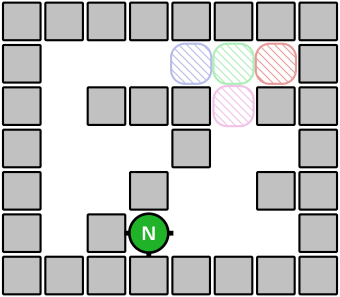
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$

Generalized moves: example



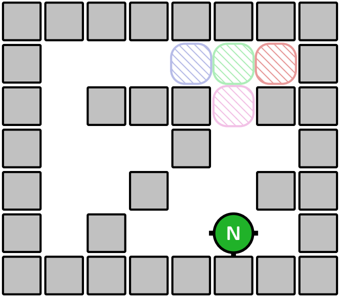
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$

Generalized moves: example



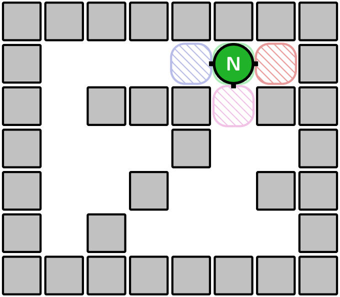
Each atom is handled independently.
The heuristic for this state is 8: 1 for **H_1**, 2 for **H_2**, 3 for **H_3** and 2 for **N**.

Generalized moves: example



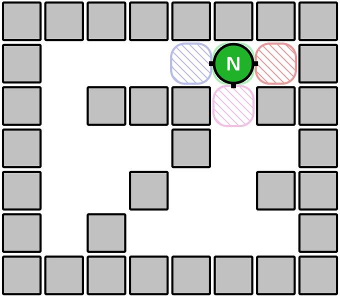
Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$ and 2 for $\mathbf{N}$.

Generalized moves: example



Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$ and 2 for **N**.

Generalized moves: example



Each atom is handled independently.
The heuristic for this state is 8: 1 for $\mathbf{H}_1$, 2 for $\mathbf{H}_2$, 3 for $\mathbf{H}_3$ and 2 for **N**.

- Distances are pre-computed before search starts by an all-pairs-shortest-paths algorithm

Pattern Databases

- Idea: map state space to a smaller, abstract state space that can be fully explored (with backward BFS from goal) and keep the optimal distances to each abstract state in a look-up table, called a PDB

Pattern Databases

- Idea: map state space to a smaller, abstract state space that can be fully explored (with backward BFS from goal) and keep the optimal distances to each abstract state in a look-up table, called a PDB
- Solution to abstract state must be a lower bound to the solution of the original state

Pattern Databases

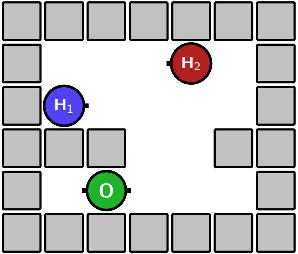
- Idea: map state space to a smaller, abstract state space that can be fully explored (with backward BFS from goal) and keep the optimal distances to each abstract state in a look-up table, called a PDB
- Solution to abstract state must be a lower bound to the solution of the original state
- Normally, in puzzles like Atomix, abstraction is done by removing some pieces

Pattern Databases

- Idea: map state space to a smaller, abstract state space that can be fully explored (with backward BFS from goal) and keep the optimal distances to each abstract state in a look-up table, called a PDB
- Solution to abstract state must be a lower bound to the solution of the original state
- Normally, in puzzles like Atomix, abstraction is done by removing some pieces
- In particular, we are interested in **additive PDBs**: if the abstract states use disjoint sets of pieces, we can add their distances and still be admissible

PDBs for Atomix: cannot remove pieces

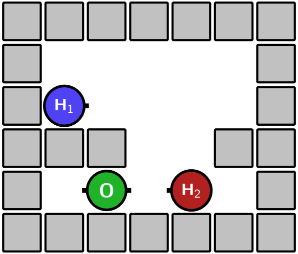
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

PDBs for Atomix: cannot remove pieces

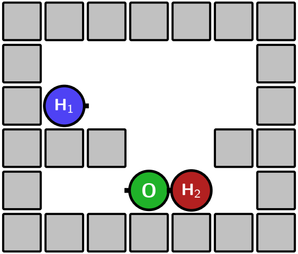
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

PDBs for Atomix: cannot remove pieces

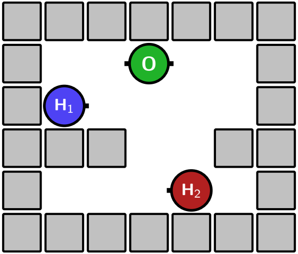
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

PDBs for Atomix: cannot remove pieces

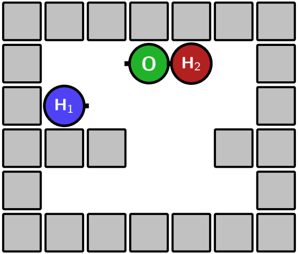
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its “cave” and meet **H_1**

PDBs for Atomix: cannot remove pieces

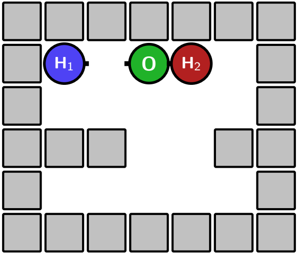
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its “cave” and meet **H_1**

PDBs for Atomix: cannot remove pieces

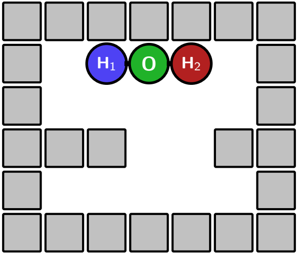
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from $\mathbf{H_2}$ to leave its “cave” and meet $\mathbf{H_1}$

PDBs for Atomix: cannot remove pieces

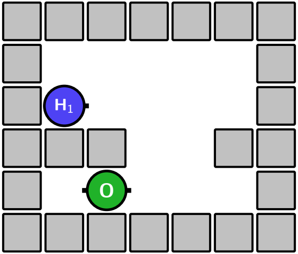
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from $\mathbf{H}_2$ to leave its "cave" and meet $\mathbf{H}_1$

PDBs for Atomix: cannot remove pieces

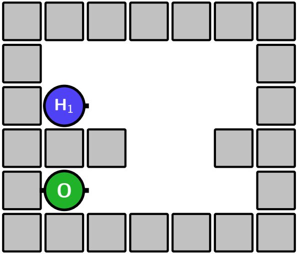
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from $\mathbf{H_2}$ to leave its "cave" and meet $\mathbf{H_1}$

PDBs for Atomix: cannot remove pieces

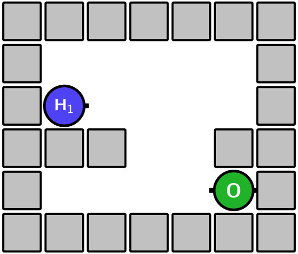
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

PDBs for Atomix: cannot remove pieces

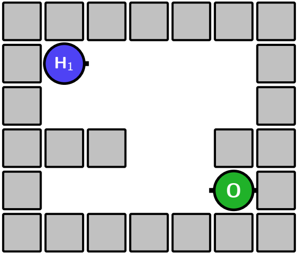
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its “cave” and meet **H_1**

PDBs for Atomix: cannot remove pieces

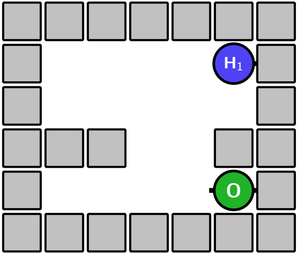
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its “cave” and meet **H_1**

PDBs for Atomix: cannot remove pieces

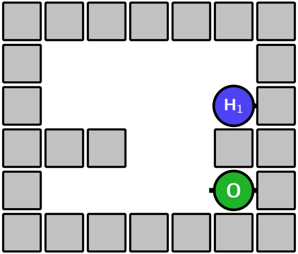
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

PDBs for Atomix: cannot remove pieces

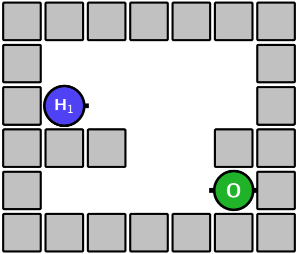
In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from $\mathbf{H_2}$ to leave its “cave” and meet $\mathbf{H_1}$

PDBs for Atomix: cannot remove pieces

In Atomix, an abstraction that only removes pieces is inadmissible:



O needs a **positive interaction** from **H_2** to leave its "cave" and meet **H_1**

Constructing PDBs for Atomix

- Using the generalized moves relaxation, atoms do not need positive interactions to reach places: they can stop anywhere they want

Constructing PDBs for Atomix

- Using the generalized moves relaxation, atoms do not need positive interactions to reach places: they can stop anywhere they want
- Removing pieces in the relaxed version is an admissible heuristic for the relaxed version

Constructing PDBs for Atomix

- Using the generalized moves relaxation, atoms do not need positive interactions to reach places: they can stop anywhere they want
- Removing pieces in the relaxed version is an admissible heuristic for the relaxed version
- An admissible heuristic for the relaxed version is also admissible for the standard version

Constructing PDBs for Atomix

- Using the generalized moves relaxation, atoms do not need positive interactions to reach places: they can stop anywhere they want
- Removing pieces in the relaxed version is an admissible heuristic for the relaxed version
- An admissible heuristic for the relaxed version is also admissible for the standard version
- A PDB on relaxed Atomix penalizes negative interactions (such as linear conflicts) between atoms within the PDB's set of pieces

Statically-partitioned PDB

- Partition the n atoms into disjoint subsets (patterns) of size k, and construct PDB for each pattern

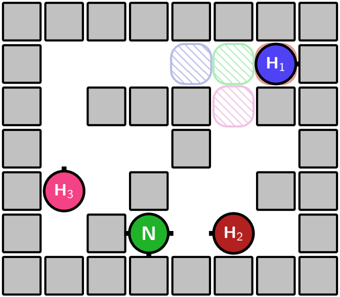
Statically-partitioned PDB

- Partition the n atoms into disjoint subsets (patterns) of size k, and construct PDB for each pattern
- Heuristic is the sum of the optimal solutions of each pattern of k atoms. This is stored in the PDB. Complexity: $O(\lfloor n/k \rfloor)$

Statically-partitioned PDB

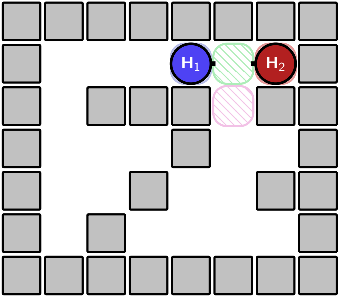
- Partition the n atoms into disjoint subsets (patterns) of size k, and construct PDB for each pattern
- Heuristic is the sum of the optimal solutions of each pattern of k atoms. This is stored in the PDB. Complexity: $O(\lfloor n/k \rfloor)$
- Chosen patterns of k of atoms stay the same during search
  - Some partitions may yield better overall lower bounds than others
  - Would be good to pick a partition that captures more conflicts within the patterns

Statically-partitioned PDB: example



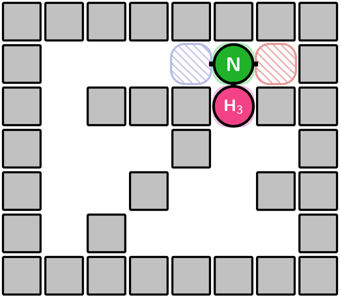
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



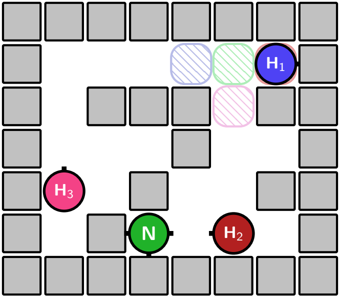
- Consider the partition with $k = 2$ $\{\mathbf{H_1}, \mathbf{H_2}\}$ and $\{\mathbf{H_3}, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H_1}, \mathbf{H_2}\}$ (BFS from goal, store distances to all possible configurations in memory)

Statically-partitioned PDB: example



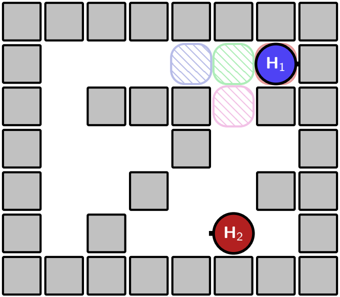
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



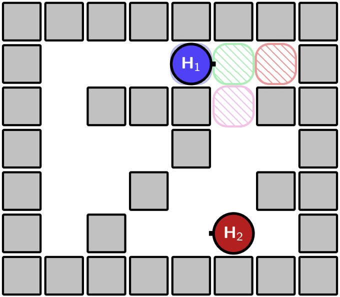
- Consider the partition with $k = 2$ $\{$**H**$_1$, **H**$_2\}$ and $\{$**H**$_3$, **N**$\}$
- Before search starts, build PDB for $\{$**H**$_1$, **H**$_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{$**H**$_3$, **N**$\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{$**H**$_1$, **H**$_2\}$ and $\{$**H**$_3$, **N**$\}$

Statically-partitioned PDB: example



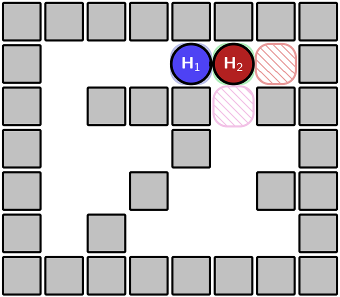
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



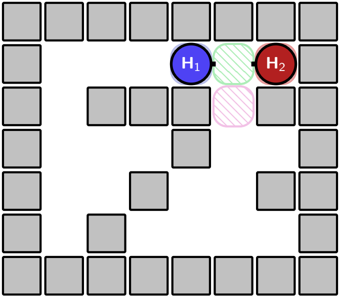
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



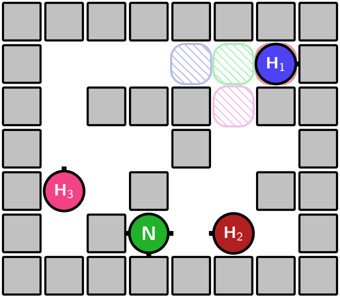
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



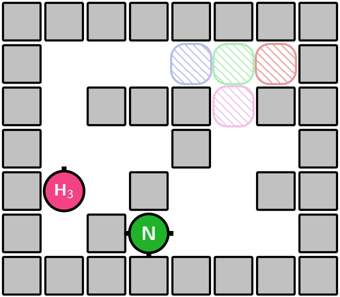
- Consider the partition with $k = 2$ $\{H_1, H_2\}$ and $\{H_3, N\}$
- Before search starts, build PDB for $\{H_1, H_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{H_3, N\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{H_1, H_2\}$ and $\{H_3, N\}$

Statically-partitioned PDB: example



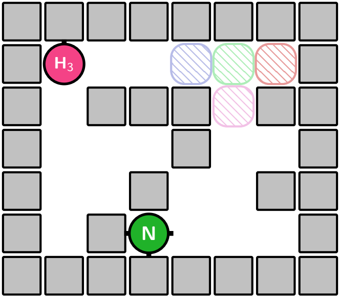
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



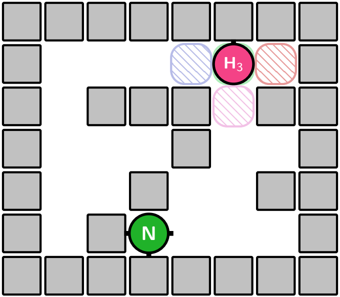
- Consider the partition with $k = 2$ $\{$**H**$_1$, **H**$_2\}$ and $\{$**H**$_3$, **N**$\}$
- Before search starts, build PDB for $\{$**H**$_1$, **H**$_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{$**H**$_3$, **N**$\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{$**H**$_1$, **H**$_2\}$ and $\{$**H**$_3$, **N**$\}$

Statically-partitioned PDB: example



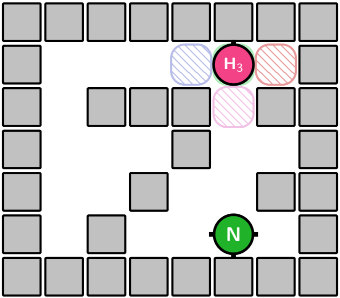
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



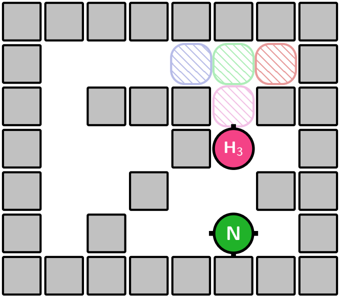
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



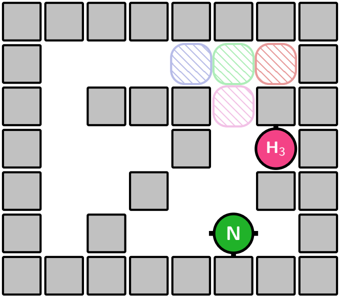
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



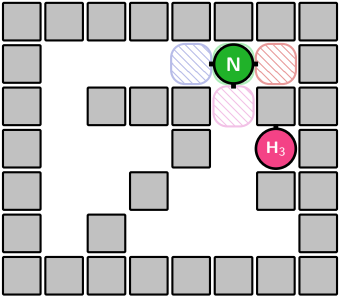
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



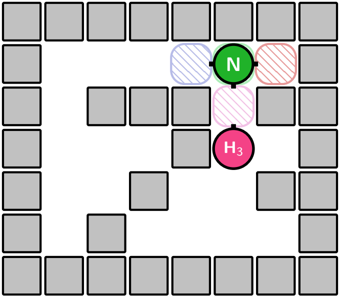
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



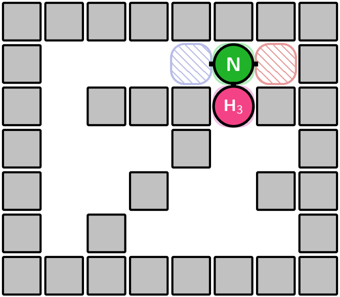
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



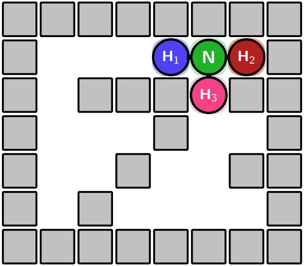
- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example



- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$

Statically-partitioned PDB: example

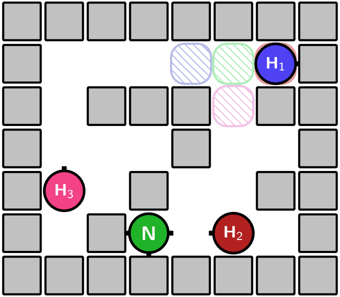


- Consider the partition with $k = 2$ $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Before search starts, build PDB for $\{\mathbf{H}_1, \mathbf{H}_2\}$ (BFS from goal, store distances to all possible configurations in memory)
- Same thing for $\{\mathbf{H}_3, \mathbf{N}\}$
- To compute heuristic of state, add heuristic of corresponding abstract states $\{\mathbf{H}_1, \mathbf{H}_2\}$ and $\{\mathbf{H}_3, \mathbf{N}\}$
- Final heuristic: 3 for $\{\mathbf{H}_1, \mathbf{H}_2\}$ and 8 for $\{\mathbf{H}_3, \mathbf{N}\}$, total 11. Recall that without PDBs it was 8

Dynamically-partitioned PDB

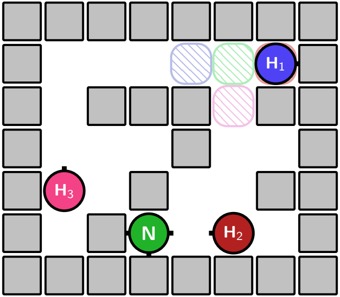
- Build PDB for all possible $\binom{n}{k}$ partitions of the n atoms
- At every heuristic call, choose the partition with maximum heuristic value
- For $k = 2$, the best heuristic is obtained by a maximum weight perfect matching

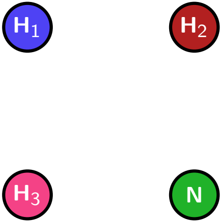
Dynamically-partitioned PDB: matching example



- Build a full graph where:

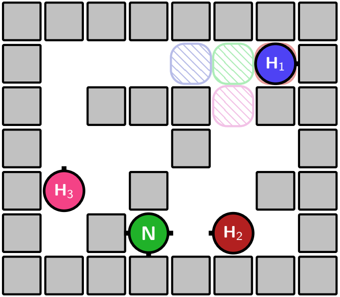
Dynamically-partitioned PDB: matching example

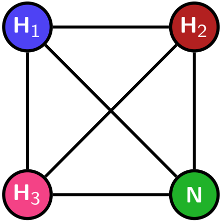




- Build a full graph where:
- Each atom is a node

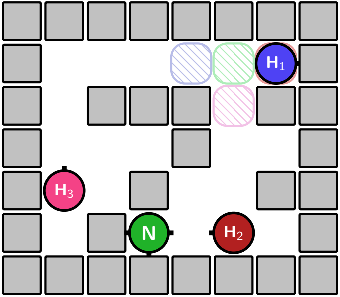
Dynamically-partitioned PDB: matching example

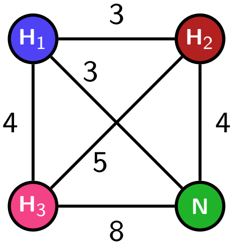




- Build a full graph where:
- Each atom is a node

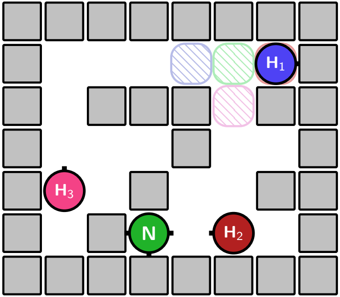
Dynamically-partitioned PDB: matching example

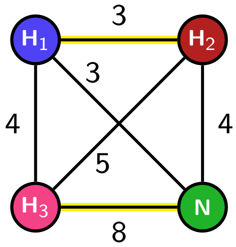




- Build a full graph where:
- Each atom is a node
- Weight of each edge is the solution of the abstract state using only the two atoms that edge connects

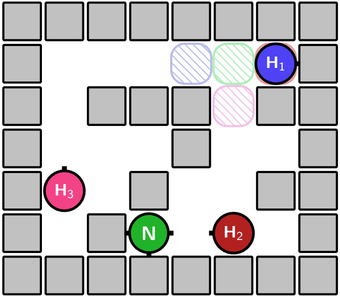
Dynamically-partitioned PDB: matching example

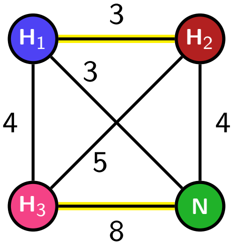




- Build a full graph where:
- Each atom is a node
- Weight of each edge is the solution of the abstract state using only the two atoms that edge connects
- Compute maximum cost perfect matching, in $O(n^3)$

Dynamically-partitioned PDB: matching example





- Build a full graph where:
- Each atom is a node
- Weight of each edge is the solution of the abstract state using only the two atoms that edge connects
- Compute maximum cost perfect matching, in $O(n^3)$
- If $k \geq 3$, matching is NP-hard: better use heuristic methods

Experimental setup

- 155 instances, averaging 10.39 atoms and 5.57 goal states per instance

Experimental setup

- 155 instances, averaging 10.39 atoms and 5.57 goal states per instance
- AMD FX-8150 1.4 GHz, 32 GB RAM

Experimental setup

- 155 instances, averaging 10.39 atoms and 5.57 goal states per instance
- AMD FX-8150 1.4 GHz, 32 GB RAM
- Tests limited to 1h execution time and 10GB memory

Experimental setup

- 155 instances, averaging 10.39 atoms and 5.57 goal states per instance
- AMD FX-8150 1.4 GHz, 32 GB RAM
- Tests limited to 1h execution time and 10GB memory
- 10 replications of each run

PDB Results

	Static $k = 3$	Dynamic $k = 2$	No PDB
# Solved	**82.8**	71	77
Avg. rel. deviation (%)	**0.58**	1.72	1.47
Avg. initial heuristic (%)	24.04	**23.39**	26.23
Time (s)	**2,952**	17,405	3,420
Nodes expanded ($\times 10^8$)	3.39	**2.39**	8.72

Comparison with Hüffner et al. (2001)

	This work (w/ static PDB)	Hüffner et al.
# Solved	**82.8**	75
Avg. rel. deviation (%)	**0.56**	1.67
Time (s)	**9,211**	12,962
Nodes expanded ($\times 10^8$)	**6.79**	47.18

Conclusions

- PDBs produce better heuristics and reduce node expansions by a factor of 2 to 3, on average

Conclusions

- PDBs produce better heuristics and reduce node expansions by a factor of 2 to 3, on average
- Static PDB yields better lower bounds than standard heuristic and in same time $O(n)$

Conclusions

- PDBs produce better heuristics and reduce node expansions by a factor of 2 to 3, on average
- Static PDB yields better lower bounds than standard heuristic and in same time $O(n)$
- Dynamic PDB yields even better lower bounds, even with smaller k, but is very slow to compute ($O(n^3)$)

Conclusions

- PDBs produce better heuristics and reduce node expansions by a factor of 2 to 3, on average
- Static PDB yields better lower bounds than standard heuristic and in same time $O(n)$
- Dynamic PDB yields even better lower bounds, even with smaller k, but is very slow to compute ($O(n^3)$)
- Static PDB solves 7.8 more instances and expands, on average, 6.94 times fewer nodes compared to previous state of the art

Questions?