CENTRO DE CIENCIAS EXATAS E TECNOLOGICAS - CCE

DEPARTAMENTO DE INFORMATICA

DISSERTATION:

ON SELECTING OF HEURISTICS FUNCTIONS FOR DOMAIN INDEPENDENT PLANNING

Marvin Abisrror Zarate

MSc Student in Computer Science

Levi Henrique Santana de Lelis

(Advisor)

Santiago Franco

(Co-Advisor)

VIÇOSA - MINAS GERAIS

MARCH - 2016

Abstract

In this dissertation we present a greedy method based on the theory of supermodular optimization for selecting a subset of heuristics functions from a large set of heuristics with the objective of reducing the running time of the search algorithms.

[Holte et al., 2006] showed that search can be faster if several smaller pattern databases are used instead of one large pattern database. We introduce a greedy method for selecting a subset of the most promising heuristicss from a large set of heuristics functions to guide the A* search algorithm. If the heuristics are consistent, our method selects a subset which is guaranteed to be near optimal with respect to the resulting A* search tree size. In addition to being consistent, if all heuristics have the same evaluation time, our subset is guaranteed to be near optimal with respect to the resulting A* running time. We implemented our method in Fast Downward and showed empirically that it produces heuristics which outperform the state of the art heuristics in the International Planning Competition benchmarks.

In this dissertation we advance and develop the approach of selection for solving different state-space problems. Namemly,

- Develop an approach for selecting a subset of heuristic functions with the goal of reducing the running time of the search algorithms employing these functions.
- Develop approaches to obtain the cardinality of the subsets of heuristics found.
- Develop a method to find a subset of heuristics from a large pool of heuristics that optimize the number of nodes expanded in the process of search.
- Use Stratified Sampling (SS) algorithm for predicting the search tree size of Iterative-Deepening A* (IDA*). We use SS as our utility function.

Contents

1 Introduction

Every problem of Artificial Intelligent (AI) can be cast as a state space problem. The state space is a set of states where each state represent a possible solution to the problem and each state is linked with other states if exists a function that goes from one state to another. In the search space there are many solutions that represent the same state, each of this solutions are called node. So, many nodes can be represented as one state. To find the solution of the problem is required the use of search algorithms such as: Depth First Search (DFS), which looks the solution of the problem traversing the search space exploring the nodes in each branch before backtracking up to find the solution. Another search algorithm is Breadth First Search (BFS), which looks for the solution exploring the neighbor nodes first, before moving to the next level of neighbors. The mentioned algorithms have the characteristic that when they do the search, they generate a larger search space, basically for two main reasons: a) Consider the total number of states to be analyzed in order to determinate if the solution is found. b) There is no guide to get to the solution. The search space that these algorithms generate are called Brute force search tree (BFST).

There are other types of algorithms called heuristic informed search, which are algorithms that requires the use of heuristics. The heuristic is the estimation of the distance for one node in the search tree to get to the near solution. The heuristic informed search generates a smaller search tree in comparison to the BFST, because the heuristic guides the search exploring the nodes that are in the solution path and prunes the nodes which are not. Also, the use of heuristics reduce the running time of the search algorithm.

There are different approaches to create heuristics, such as: Pattern Databases (PDBs), Neural Network, and Genetic Algorithm. These systems that create heuristics receive the name of Heuristics Generators. And one of the approaches that have showed most successfull results in heuristic generation is the PDBs, which is memory-based heuristic functions obtained by abstracting away certain problem variables, so that the remaining problem (”pattern”) is small enough to be solved optimally for every state by blind exhaustive search. The results stored in a table, represent a PDB for the original problem. The abstraction of the search space gives an admissible heuristic function, mapping states to lower bounds.

Exists many ways to take advantage of all the heuristics that can be created, for example: [Holte et al., 2006] showed that search can be faster if several smaller pattern databases are used instead of one large pattern database. In addition [Domshlak et al., 2010] and [Tolpin et al., 2013] results showed that evaluating the heuristic lazily, only when they are essensial to a decision to be

made in the search process is worthy in comparison to take the maximum of the set of heuristics. Then, using all the heuristics do not guarantees to solve the major number of problems in a limit time.

Finally, the objective of this dissertation is to develop meta-reasoning approaches for selecting heuristics functions from a large set of heuristics with the goal of reducing the running time of search algorithm employing these functions.

2 Background

A SAS^{+} *planning task* [Bäckström and Nebel, 1995] is a 4 tuple $\nabla = \{V, O, I, G\}$. V is a set of *state variables.* Each variable $v \in V$ is associated with a finite domain of possible D_v. A state is an assignment of a value to every $v \in V$. The set of possible states, denoted V, is therefore $D_{v_1} \times ... \times D_{v_2}$. O is a set of operators, where each operator $o \in O$ is triple $\{pre_o, post_o, cost_o\}$ specifying the preconditions, postconditions (effects), and non-negative cost of o. pre_o *and* $post_o$ are assignments of values to subsets os variables, V_{pre_o} *and* V_{post_o}, respectively. Operator o is applicable to state s if s and pre_o agree on the assignment of values to variables in V_{pre_o}. The effect of o, when applied to s, is to set the variables in V_{post_o} to the values specified in $post_o$ and to set all other variables to the value they have in s. G is the goal condition, an assignment of values to a subset of variables, V_G. A state is a goal state if it and G agree on the assignment of values to the variable in V_G. I is the initial state, and the planning task, ∇, is to find an optimal (least-cost) sequence of operators leading from I to a goal state. We denote the optimal solution cost of ∇ as C^*

The state space problem illustrated in the figure 1 is a game that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, particularly the smaller $8-$puzzle. If the size is 3×3 tiles, the puzzle is called the 8 puzzle or $9-$puzzle, and if 4×4 tiles, the puzzle is called the 15-puzzle or 16-puzzle named, respectively, for the number of tiles and the number of spaces. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

The legal operators are to slide any tile that is horizontally or vertically adjacent to the blank into the blank position. The problem is to rearrange the tiles from some random initial configuration into a particular desired goal configuration. The $8-$puzzle contains 181,440 reachable states, the $15-$puzzle contains about 10^{13} reachable states, and the $24-$puzzle contains almost 10^{25} states.

Initial

4	1	2
8		3
5	7	6

Goal

1	2	3
4	5	6
7	8	

Figure 1: The left tile−puzzle is the initial distribution of tiles and the right tile−puzzle is the goal distribution of tiles. Each one represent a State.

Instead of using an algorithm of Brute force search that will analyze all the possible solutions. We can obtain heuristics from the problem of the slide tile puzzle that will help us to solve the problem.

3 Heuristics

State−space algorithms, such as A* [Hart and Raphael, 1968], are important in many AI applications. A* uses the $f(s) = g(s) + h(s)$ cost function to guide its search. Here, $g(s)$ is the cost of the path from the start state s, and $h(s)$ is the estimated cost−to−go from s to a gial; $h(.)$ is known as the heuristic function. The heuristic is the mathematical concept that represent to the estimate distance from the node s to the nearest goal state.

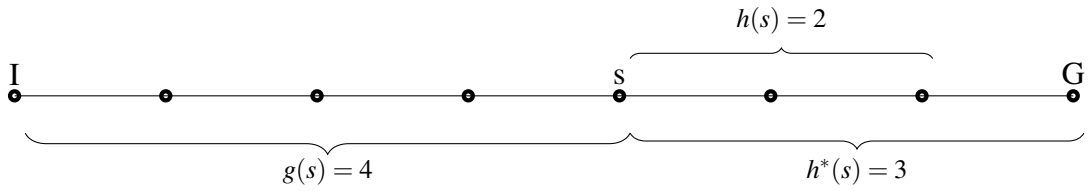


Figure 2: Heuristic Search: *I*: Initial State, *s*: Some Sate, *G*: Goal State

In the figure 2 the optimal distance from the Initial State I to the state s is 4 and represented by $g(s)$. The $h^*(s)$ represent the optimal distance from s to the Goal State G. And the $h(s)$ is the estimation distance from s to G.

The heuristics can be obtained from each state of the problem. For example, for the problem of the 8−tile−puzzle figure 1 we can get two heuristics.

3.1 Out of place (O.P)

Counts the number of objects out of place.

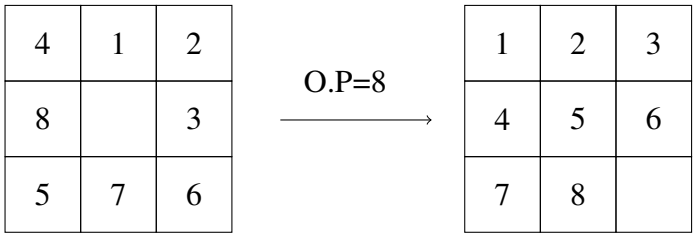


Figure 3: Out of place heuristic

The tiles numbered with 4, 1, 2, 3, 6, 7, 5, 8, and 4 are out of place then each object count as 1 and the sum would be 8.

3.2 Manhatham Distance (M.D)

Counts the minimum number of operations to get to the goal state.

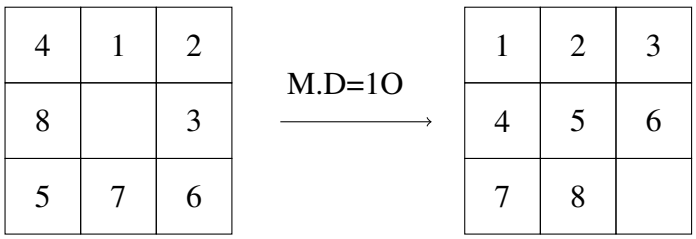


Figure 4: Manhatham distance heuristic

The tile 4 count 1 to get to the goal position. The tile 1 count 1 to get to the goal position. The tile 2 count 1 to get to the goal position. The tile 3 count 1 to get to the goal position. The tile 6 count 1 to get to the goal position. The tile 7 count 1 to get to the goal position. The tile 5 count 1 to get to the goal position. The tile 8 count 1 to get to the goal position. Then the sum would be 10.

In order to solve the problem, we get the heuristics, which are information from the problem to solve the problem. Exists systems that can create heuristics for each problem. Those systems are called Heuristic Generators.

4 Heuristic Generators

Heuristic Generators works by creating abstractions of the original problem spaces. There are different ways to abstract the problem space such as:

4.1 Pattern Database (PDB)

It's obtained by abstracting away certain problem variables, so that the remaining problem ("pattern") is small enough to be solved optimally for every state by blind exhaustive search. The results stored in a table, represent a PDB for the original problem. The abstraction of the search space gives an admissible heuristic function, mapping states to lower bounds.

4.2 Neural Network

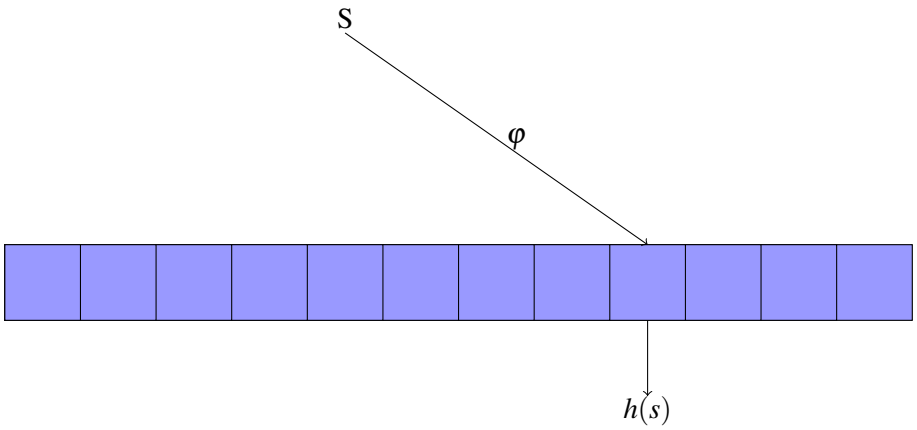
4.3 Genetic Algorithm

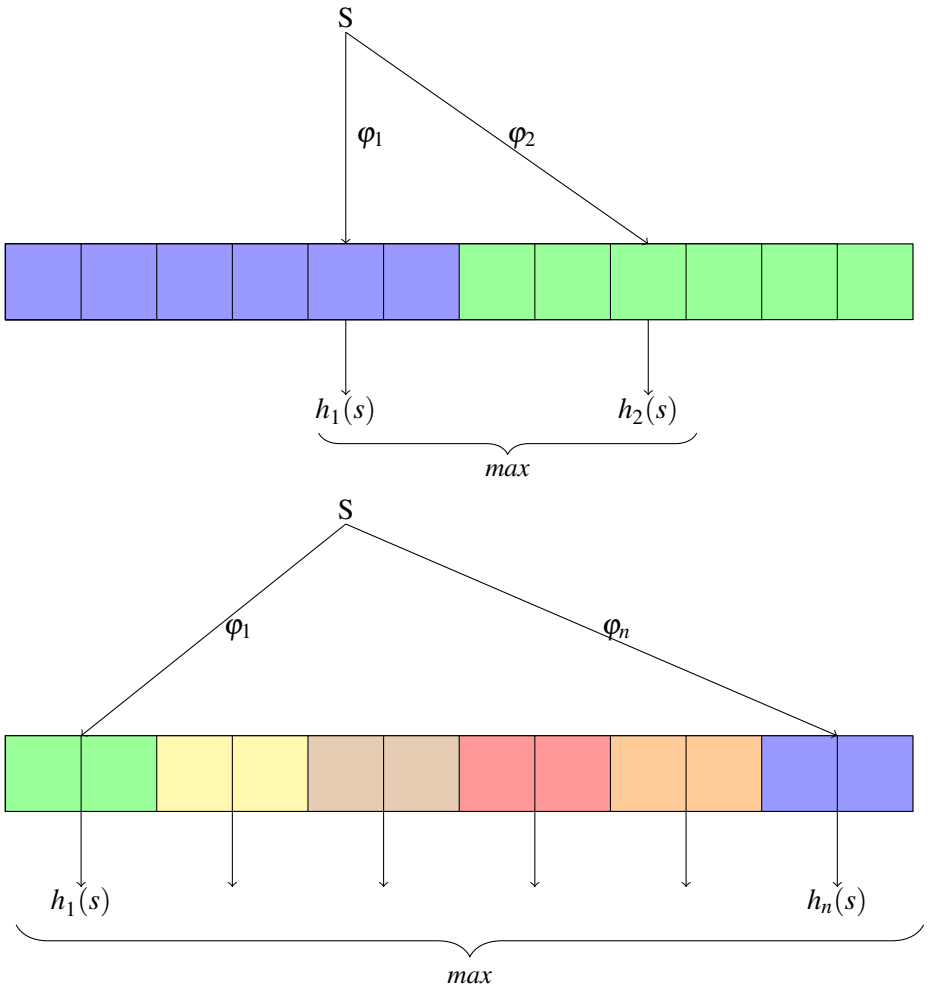
5 Take advantage of Heuristics

The heuristics generators can create hundreds or even thousand of heuristics. In fact, exists different ways to take advantage of those heuristics. For example: If we want to use all the heuristics created by the heuristic generator that might be thousands of them, would not be a good idea to use all of them because the main problem involved would be the time to evaluate each heuristic in the search tree, it could take too much time.

One way to take advantage of heuristics would be to take the maximum of the set of heuristics. For example, using three different of heuristics $h1, h2$ and $\max(h1, h2)$. Heuristic $h1$ and $h2$ are based on domain abstractions.

6 Number of heuristics to create.





7 Heuristic Subset

$|\zeta| = 1000$ heuristics, N = 100

$$\binom{1000}{100} = 10^{138} possibilities$$

8 Properties

8.1 Monotonic Function

8.2 Submodularity

9 Greedy Heuristic Selection (GHS)

10 Stratified Sampling (SS)

10.1 Type System

11 Experiments

12 Conclusions

Define the problem using examples

The new approach for selecting a subset of heuristics functions for domain-independent planning has two main objectives: First, make a selection of heuristics from a large set of heuristics with the goal of reducing the running time of a search algorithm employing the subset functions. Second, find out if the prediction of Stratified Sampling (SS) might be helpful in selecting a subset of heuristics to guide the A* search.

Maybe writing [Krause and Golovin, 2012] should demostrated something. Cite by holte [Holte et al., 2006] and this. Another cite [Xu et al., 2014] what about this. Another cite [Krause and Guestrin, 2007] what about this. Another cite by bernand nebel [Bäckström and Nebel, 1995] what about this.

In order to achieve the first objective we present The Greedy Algorithm, which provides a good approximation to the optimal solution of the NP-hard optimization problem [Krause and Golovin, 2012].

In order to achieve the second objective we use the *relative unsigned error* to probe the accuracy of the predictions of SS with respect to IDA*. We know that SS does not make even reasonable predictions for the number of nodes expanded by A*. Nevertheless, even though SS produces poor predictions for the number of nodes expanded by A*, we would like to verify whether these predictions can be helpful in selecting a subset of heuristics to guide the A* search.

This report have three sections, the first section is the introduction, the second is the experiment 1 which contains four main tables showing the results of the *relative unsigned error.* and the last section in the conclusion.

13 Comparison of SS with IDA*

Stratification Sampling is an algorithm that estimates the number of nodes expanded performed by heuristic search algorithm seeking solutions in state space. We apply SS to predict the number of nodes expanded by IDA* in a given f-layer when using a consistent heuristics.

We first ran IDA* for Fast-Downward benchmark for optimal domains. Our evaluation metric is coverage, *i.e.,* number of problems solved within 24 hours time limit. We note that in 24 hours non all the instances for a specific domain using a consistent heuristic can be solved. Afterwards, run SS using as a threshold the f-layer for each instance of each domain, this process is executed using different number of probes *i.e.,* 1, 10, 100, 1000, and 5000.

In our experiment 1, prediction accuracy is measured in terms of the *Relative Unsigned Error* (ss-err), which is calculated as:

$$\frac{\sum_{s \in PI} \frac{Pred(s,d) - R(s,d)}{R(s,d)}}{|PI|}$$

Where *PI* is the set of problem instances, *Pred(s, d)* and *R(s, d)* are the predicted and actual number of nodes expanded by IDA* for start state s and cost bound d. A perfect score according to this measure is 0.000.

The heuristics used for this experiment 1 were: hmax, ipdb, lmcut, and merge_and_shrink. There are 4 tables, each table shows the results running IDA* and SS using one consistent heuristic. The first column represent the optimal domains for Fast-Downward benchmark. The remaining 10 columns shows the 5 different probes *i.e.,* 1, 10, 100, 1000, and 5000. Each probe has two columns which represent the ss-err and the ss-time. The last two columns are the information for IDA* which represent the average value of the number of nodes expanded and the average time respectively. The text ”—” means that IDA* could not solve the problems, consequently there are not results for SS.

In the Table **??** we can see that there are seven domains which IDA* could not solve any instance in 24 hours. The ss-err decrease for each domain according the number of probes increases. The domains that have the perfect score are: openstacks-opt11-strips, parcprinter-opt11-strips.

Table 1: Experiment 1 - Comparison using hmax heuristic

	hmax												
	1		10		100		1000		5000				
Domain	error	time	error	time	error	time	error	time	error	time	ida*	ida*-time	n
barman-opt11	0.60	0.06	0.45	0.32	0.20	3.21	0.07	32.57	0.04	214.59	8835990.00	6016.38	20
blocks	0.42	0.02	0.17	0.10	0.06	1.06	0.03	10.86	0.01	65.97	28510300.00	3030.97	35
elevators-opt08	0.67	1.61	0.48	11.13	0.21	110.38	0.13	1140.05	0.48	3012.95	923397.00	4795.09	30
elevators-opt11	0.84	1.40	0.42	9.85	0.23	96.37	0.13	994.33	0.14	4223.73	966309.00	4759.72	20
floortile-opt11	2.02	0.01	0.62	0.07	0.40	0.69	0.14	6.93	0.11	36.60	30522300.00	3919.72	2
nomystery-opt11	0.53	0.07	0.26	0.38	0.07	3.63	0.03	36.35	0.01	181.03	6565740.00	3256.86	20
openstacks-adl	—	—	—	—	—	—	—	—	—	—	—	—	—
openstacks-opt08	0.58	82.20	0.04	800.37	0.10	1260.88	0.10	12368.90	0.22	9594.93	73087.30	2669.55	30
openstacks-opt11	0.03	94.79	0.03	774.86	0.10	991.57	0.10	10148.40	0.24	8779.80	62942.40	3156.36	20
parcprinter-opt11	0.00	0.01	0.00	0.04	0.00	0.35	0.00	3.48	0.00	17.29	1.00	0.00	20
parking-opt11	0.17	1.79	0.04	11.36	0.01	114.28	0.00	1196.83	0.00	5835.03	374925.00	5607.50	20
pegsol-opt11	0.17	0.01	0.04	0.04	0.02	0.37	0.01	3.69	0.00	17.88	68763.70	5.00	20
scanalyzer-opt11	0.43	3.13	0.25	28.79	18.63	273.74	0.02	3033.06	0.04	10021.00	8257850.00	4808.75	20
sokoban-opt08	0.35	0.27	0.22	1.95	0.09	20.24	0.05	214.01	0.03	965.40	2657890.00	3385.95	30
sokoban-opt11	0.41	0.31	0.26	2.00	0.11	21.42	0.05	222.47	0.04	1056.61	3118530.00	3932.69	20
tidybot-opt11	300.86	4.40	1072.40	26.48	5.88	238.76	0.01	2747.10	0.04	11572.90	431336.00	5465.62	20
transport-opt08	0.55	0.33	0.36	2.57	0.27	23.11	0.13	236.72	0.10	1363.04	1462640.00	957.41	27
transport-opt11	0.63	0.09	0.54	0.61	0.24	5.89	0.15	59.37	0.11	290.31	2622880.00	2253.51	20
visitall-opt11	0.12	0.00	0.04	0.05	0.01	0.56	0.00	5.77	0.00	28.07	71032400.00	3704.78	20
woodworking-opt08	0.89	0.16	0.57	1.44	0.35	13.94	0.13	140.83	0.07	685.86	4170080.00	4055.03	30
woodworking-opt08	1.28	0.15	0.69	1.33	0.27	13.21	0.17	130.82	0.07	664.08	5139070.00	4944.76	20

References

Robert C Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170(16):1123–1136, 2006.

Carmel Domshlak, Erez Karpas, and Shaul Markovitch. To max or not to max: Online learning for speeding up optimal planning. In *AAAI*, 2010.

David Tolpin, Tal Beja, Solomon Eyal Shimony, Ariel Felner, and Erez Karpas. Towards rational deployment of multiple heuristics in a*. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 674–680. AAAI Press, 2013.

Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

N. J.; Hart, P. E.; Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC*, 4(2):100–107, 1968.

Table 2: Experiment 1 - Comparison using hmax heuristic

	hmax												
			ss-error					time					
Domain	IDA*	time	1	10	100	1000	5000	1	10	100	1000	50000	n
barman-opt11	8835990.00	6016.38	0.60	0.45	0.20	0.07	0.04	0.06	0.32	3.21	32.57	214.59	20
blocks	28510300.00	3030.97	0.42	0.17	0.06	0.03	0.01	0.02	0.10	1.06	10.86	65.97	35
elevators-opt08	1628240.00	8455.25	0.67	0.48	0.21	0.13	0.09	1.61	11.13	110.38	1140.05	5312.77	30
elevators-opt11	1012570.00	4987.57	0.84	0.42	0.23	0.13	0.10	1.40	9.85	96.37	994.33	4425.93	20
floortile-opt11	30522300.00	3919.72	2.02	0.62	0.40	0.14	0.11	0.01	0.07	0.69	6.93	36.60	2
nomystery-opt11	6565740.00	3256.86	0.53	0.26	0.07	0.03	0.01	0.07	0.38	3.63	36.35	181.03	20
openstacks-adl	—	—	—	—	—	—	—	—	—	—	—	—	—
openstacks-opt08	89953.60	3285.60	0.58	0.04	0.04	0.04	0.04	82.20	800.37	1344.94	13193.50	11809.10	30
openstacks-opt11	80108.50	4017.19	0.03	0.03	0.03	0.03	0.03	94.79	774.86	1067.84	10929.00	11174.30	20
parcprinter-opt11	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.04	0.35	3.48	17.29	20
parking-opt11	374925.00	5607.50	0.17	0.04	0.01	0.00	0.00	1.79	11.36	114.28	1196.83	5835.03	20
pegsol-opt11	68763.70	5.00	0.17	0.04	0.02	0.01	0.00	0.01	0.04	0.37	3.69	17.88	20
scanalyzer-opt11	8449890.00	4920.58	0.43	0.25	18.63	0.02	0.01	3.13	28.79	273.74	3033.06	10254.00	20
sokoban-opt08	2657890.00	3385.95	0.35	0.22	0.09	0.05	0.03	0.27	1.95	20.24	214.01	965.40	30
sokoban-opt11	3118530.00	3932.69	0.41	0.26	0.11	0.05	0.04	0.31	2.00	21.42	222.47	1056.61	20
tidybot-opt11	444473.00	5632.08	300.86	1072.40	5.88	0.01	0.01	4.40	26.48	238.76	2747.10	11925.40	20
transport-opt08	1462640.00	957.41	0.55	0.36	0.27	0.13	0.10	0.33	2.57	23.11	236.72	1363.04	27
transport-opt11	2622880.00	2253.51	0.63	0.54	0.24	0.15	0.11	0.09	0.61	5.89	59.37	290.31	20
visitall-opt11	71032400.00	3704.78	0.12	0.04	0.01	0.00	0.00	0.00	0.05	0.56	5.77	28.07	20
woodworking-opt08	4170080.00	4055.03	0.89	0.57	0.35	0.13	0.07	0.16	1.44	13.94	140.83	685.86	30
woodworking-opt08	5139070.00	4944.76	1.28	0.69	0.27	0.17	0.07	0.15	1.33	13.21	130.82	664.08	20

Table 3: Poor prediction of SS against A* ipdb

Experiment 2: Using ipdb heuristic - 500 in gapdb_deep			
Domain	A*	ss error	n
blocks	2.07455e+06	9.20794e+31	18
barman-opt11-strips	1.71877e+07	5.84137e+32	4
elevators-opt08-strips	1.39911e+07	9.2879e+23	7
elevators-opt11-strips	1.83142e+07	1.3003e+24	5
floortile-opt11-strips	1.40015e+07	7.52158e+16	4
nomystery-opt11-strips	40169.7	1.15599e+34	9
openstacks-opt08-strips	1874.5	999720	2
parcprinter-opt11-strips	1157	2.56184e+22	3
scanalyzer-opt11-strips	337894	3.71451e+32	3
sokoban-opt08-strips	1136.25	3.48709e+08	4
sokoban-opt11-strips	861	0.938381	1
transport-opt08-strips	755974	1.90383e+39	5
transport-opt11-strips	1.88894e+06	2.91167e+38	2
visitall-opt11-strips	8.12101e+06	3.18043e+43	8
woodworking-opt08-strips	1.50059e+06	8.65693e+17	7
woodworking-opt11-strips	4.81226e+06	3.02819e+18	2

Table 4: Poor prediction of SS against A* lmcut

Experiment 2: Using lmcut heuristic - 500 in gapdb_deep			
Domain	A*	ss error	n
blocks	2.39089e+06	2.68279e+31	18
barman-opt11-strips	7.44986e+06	9.16154e+28	4
elevators-opt08-strips	1.17502e+07	3.7338e+19	7
elevators-opt11-strips	1.5278e+07	5.22719e+19	5
floortile-opt11-strips	702435	6.21105e+10	4
nomystery-opt11-strips	267100	1.03783e+26	9
openstacks-opt08-strips	1874.5	953648	2
parcprinter-opt11-strips	1363.67	2.33125e+21	3
scanalyzer-opt11-strips	334747	7.55436e+30	3
sokoban-opt08-strips	1000.75	2.27145e+08	4
sokoban-opt11-strips	861	0.938381	1
transport-opt08-strips	594665	4.60306e+24	5
transport-opt11-strips	1.48569e+06	1.15083e+25	2
visitall-opt11-strips	8.1205e+06	3.18043e+43	8
woodworking-opt08-strips	1.49993e+06	9.17535e+17	7
woodworking-opt11-strips	4.81236e+06	3.2025e+18	2

Table 5: Poor prediction of SS against A* mands

Experiment 2: Using mands heuristic - 500 in gapdb_deep			
Domain	A*	ss error	n
blocks	2.96963e+06	1.01528e+31	18
barman-opt11-strips	2.67042e+07	2.08212e+36	4
elevators-opt08-strips	1.58876e+07	6.50833e+26	7
elevators-opt11-strips	2.0719e+07	9.11162e+26	5
floortile-opt11-strips	3.26068e+07	7.36763e+16	4
nomystery-opt11-strips	8236	2.02873e+26	9
openstacks-opt08-strips	1874.5	299017	2
parcprinter-opt11-strips	766.333	6.35555e+20	3
scanalyzer-opt11-strips	337893	1.76874e+29	3
sokoban-opt08-strips	602.5	4.08107e+08	4
sokoban-opt11-strips	861	0.938381	1
transport-opt08-strips	741293	2.72958e+37	5
transport-opt11-strips	1.85225e+06	6.82396e+37	2
visitall-opt11-strips	8.121e+06	3.22741e+43	8
woodworking-opt08-strips	5.48249e+06	1.21766e+18	7
woodworking-opt11-strips	1.77967e+07	4.26012e+18	2

Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012.

Haifeng Xu, Fei Fang, Albert Xin Jiang, Vincent Conitzer, Shaddin Dughmi, and Milind Tambe. Solving zero-sum security games in discretized spatio-temporal domains. *Proceedings of the 28th Conference on Artificial Intelligence (AAAI 2014), Qubec, Canada*, 2014.

Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. *AAAI*, 7:1650–1654, 2007.