Improved Airport Ground Traffic Control with Domain-Dependent Heuristics

Augusto B. Corrêa*, André G. Pereira†, Marcus Ritt‡
Institute of Informatics
Federal University of Rio Grande do Sul, Brazil
{abcorrea*,agpereira†,marcus.ritt‡}@inf.ufrgs.br

***Abstract*—In this paper we study the application of a domain-dependent heuristic to airport ground traffic control. We consider two variants of the problem. In the first, proposed for the International Planning Competition in 2004, the in-bound and out-bound airplanes have fixed parking and take-off positions. In the second, more realistic variant a controller can assign dynamically for each airplane the runway for take-off or the parking position, such that the total movement of planes at the airport is minimized. We are particularly interested in the second variant, which has an implicitly defined goal state where multiple states could satisfy the goal condition, and the impact of this fact on domain-independent and domain-dependent heuristics. We compare domain-independent heuristics in the Fast Downward planner on this domain to a domain-dependent heuristic.**

***Keywords*-Airport planning, Single-agent search, Heuristic search, Domain-specific heuristics, A*.**

I. Introduction

One of the planning problems that arises in airports is the ground traffic control problem, which is to define the paths out-bound airplanes have to take from their parking position to the runway for take-off, or the paths in-bound airplanes have to take after landing until getting to a parking position. The main objective is to find a plan that achieves this goal most economically, i.e. with the least movement of the airplanes, respecting restrictions on the movement coming from concerns about security, for example the minimum distance between airplanes.

A realistic version of the airport traffic control problem has been introduced in the International Planning Competition (IPC) in 2004 [1]. In this version the airport is modeled as a directed graph, with designated parking and take-off positions. All airplanes have an orientation and other airplanes must keep a minimum distance from the rear of an airplane with a running engine.

There are several versions of the airport ground traffic control problem. In the simplest version, all routes for in- or out-bound airplanes are fixed. This is often done to simplify the decision making of the ground traffic controller. However, even this version is NP-complete [2]. If the routes are not prefixed, deciding if a polynomially bounded plan or any feasible plan exists is PSPACE-complete, even when the goal positions of the airplanes are known [3]. This is one of the reasons that makes the airport domain one of the most difficult domains in the IPC. Four versions have been introduced in IPC 2004 and they differ in the modeling of time. The simplest version is non-temporal. There also exist temporal versions with and without time windows. In practice the goal is to minimize the *makespan* of an instance, i.e. the total time or number of actions until all airplanes reach their goal positions.

In this paper we are interested in solving the non-temporal, sequential version of the problem optimally, i.e. we search for plans which minimize the total number of actions. In the instances of IPC 2004 the goal positions of the airplanes (a runway for out-bound airplanes, and a parking position for in-bound airplanes) are fixed. Here we are interested in a variant of the problem that permits free goal positions, such that in-bound airplanes can choose any of the available parking positions and out-bound airplanes can take off at several available runways. This variant is more realistic since, for example, the parking position of an airplane may change after touchdown. Besides being a more realistic variant, this change makes the problem also more difficult to solve, since for a_i in-bound airplanes, p parking positions, a_o out-bound airplanes and r runways the number of goal states increases from 1 to $\binom{p}{a_i} r^{a_o}$.

Another motivation to study this variant is that it has several similarities with Sokoban, a well-known hard problem [4], including a large number of goal states, the existence of linear conflicts (i.e. pairs of airplanes that mutually block their independent shortest path to the corresponding goal positions), and (usually) instances with a regular, structured geometry, which may be decomposed into several independently solvable subproblems. Thus we are interested in evaluating if techniques that work in Sokoban are effective for this problem.

The techniques used by solvers for airport ground traffic problems usually are domain-independent. These techniques do not take into consideration particularities of the problem, and try to solve it only by general strategies on the problem instances. A domain-dependent technique may reach better results compared to domain-independent techniques. On the airport domain, for example, a domain-dependent technique could use the distance from each segment to an airplane or the information of the structure of parking and runway positions.

Pattern database based techniques are one of the most effective domain-independent heuristics for solving this problem [5], [6]. A pattern database heuristic solves an abstraction of the concrete problem, and stores the shortest distance to the closest abstract goal state for all abstract states in a look-up table. For solving the concrete problem, querying the

look-up table for the abstraction which corresponds to the concrete state gives a valid lower bound on the distance to the shortest concrete goal states, and can be used as an admissible heuristic.

In the next section, we define the problem formally, and discuss related work. In Section III we propose two domain-dependent heuristic functions, based on shortest paths and matchings for the problem. Computational experiments are reported and discussed in Section IV. We conclude and point out some possible future research in Section V.

II. Preliminaries

Airport Ground Traffic Control

An airport is defined by a set of segments $\mathcal{S}$. Each segment $s \in \mathcal{S}$ can be either a taxiway, a parking position, or a runway for landing and take-off, and has a type $t(s) \in T = \{taxi, park, runway\}$. A position of an airplane is defined by its current segment and its direction. The direction defines to which end of the segment the airplane is oriented, and restricts the possible successor segments. Thus, an airport can be modeled by a directed graph $G = (V, A)$, with vertex set $V = \mathcal{S} \times D$, where $D = \{\textit{north}, \textit{south}\}$ represents the possible directions of an airplane, and arc set A, which defines the possible successor segments and directions for a given current segment and direction.

An instance of the ground traffic control problem is defined by a set of airplanes P, with positions $p : P \to V$ and goal segments $g : P \to \mathcal{S}$. The type of a goal segment $g(a)$ for an airplaine $a \in P$ must be either *park* for in-bound airplanes or *runway* for out-bound airplanes. An airplane can be either *moving*, *parked* or *airborne*, defined by its current mode $m(a) \in M = \{\textit{parked}, \textit{moving}, \textit{airborne}\}$.

Mode *parked* is only permitted if the current segment of the airplane has type *park*. A *parked* airplane can start up its engines to enter mode *moving*. Only in this mode an airplane can change its current segment and direction (p, d) to another segment and direction (p', d') if $((p, d), (p', d')) \in A$. If an airplane is in mode *moving* at a segment of type *park* it can similarly shut down its engines to enter mode *parked*. Mode *airborne* is only valid for segments of type *runway*. An airplane in mode *moving* at a runway can take off to switch to mode *airborne*. There is no operator for landing. All operators in the airport domain have unit cost.

A *moving* airplane at position $v \in V$ will block a set of segments $B(v) \subseteq V$ due to the engine exhaust of the airplane. The blocked segments depend on the orientation of the airplane and its size (larger airplanes may block more segments). For current positions p and modes m, segments $B(p, m) = \cup_{a \in P^g} B(p(a))$ are blocked, where $P^g(m) = \{a \in P \mid m(a) \neq \textit{airborne}\}$ is the set of airplanes on the ground. Let $\mathcal{S}^g(p, m) = \{s \mid (s, d) = p(a) \text{ for some } a \in P^g(m)\}$ be the segments occupied by airplanes on the ground. The current positions p are valid, if on each segment there is a most one airplane of type *moving* or *parked*, i.e. $|\mathcal{S}^g(p, m)| = |P^g(m)|$, and no airplane on the ground is at a blocked segment, i.e. $\mathcal{S}^g(p, m) \cap B(p, m) = \emptyset$.

The standard airport domain with fixed goal positions can be modeled as a weighted state space problem $P = (S, O, s, T, w)$, with a set of states S, a set of operators O, an initial state $s \in S$, a set of goal states $T \subseteq S$ and a cost function $w : O \to \mathsf{R}$, as follows. The set of states $S \subseteq (V \times M)^{|P|}$ consists of all feasible positions and modes of the airplanes. The initial state $s_0 = (p_0, m_0)$ is given by feasible initial positions p_0 and modes m_0. Similarly the only goal state is $T = \{s_f\}$ with $s_f = (p_f, m_f)$ given by feasible final positions p_f and modes m_f. The set of operators consists of starting engines, moving, taking off, or parking, as described above, and all actions have unit cost, i.e. $w(o) = 1$ for all operators $o \in O$. The alternative definition of the problem with free parking positions can be modeled in a similar way. In this case each airplane has only a mode that can be either *parked* or *airborne* as a goal. The aim of this definition is to simulate a situation where a controller could assign dynamically for each airplane the segment for taking off or parking. This defines a partial assignment to each airplane, and thus multiple goal states can satisfy the goal condition.

Standard Set of Instances

The standard set of instances of the airport domain represents real-world conditions of an airport ground traffic control problem. There are four versions of the domain: *non-temporal*, *temporal*, *temporal-timewindows*, and *temporal-timewindows-compiled*. The last three versions include time constraints, as for example the time to move across a segment and time windows where a segment will be blocked. We address the first version which is the most widely researched in the literature.

In the standard set, for each instance there is a unique explicitly defined goal state. The goal state is defined by a set of two predicates `(is-parked ?a ?s)` and `(airborne ?a ?s)`, where `a` is an airplane and `s` is a segment. Each predicate defines the segment that the airplane must reach to achieve the goal condition. There are two actions, `park` and `takeoff`, that make the predicates true. Thus, the transformation from fixed to free goal positions consists of removing from the `is-parked` and `airborne` predicates the segment definition. In practice, the predicate `(is-parked ?a ?s)` has been changed to `(is-parked ?a)`. A similar change has been applied to the predicate `airborne`. We did not add any other action to the domains and just adapted it to make it consistent. The resulting definition produces an implicit defined goal state, where usually multiple states will satisfy the goal condition.

The standard set from IPC-4 has 50 instances. There are five types of airports with an increasing number of segments. The two largest airport types are realistic encodings of the Munich airport. The smallest airport has 17 segments, the largest one has 457. The maximum number of planes is 15. In 31 of the 50 instances there are more airplanes which are out-bound than in-bound, and that the number of goal segments for out-bound airplanes is usually small, and never more than four. When considering the implicit defined goal state, the effect of multiple goal states in the first three types of airports is limited.

The situation changes in the more realistic airports of type 4 and 5. In these cases we have still a small number of 1 to 4 of in-bound airplanes, but 30 possible parking positions, and thus up to $\binom{30}{4}$ possible goal states for the in-bound airplanes.

Related Work

General traffic control problems have been studied by Hatzack and Nebel [2]. They proposed the only domain-dependent heuristic solution for the problem we are aware of. They also have showed that the problem with fixed routes is equivalent to a job shop scheduling problem which has to satisfy blocking conditions without swaps. The reduction to job shop scheduling also shows that the problem with fixed paths still remains NP-complete.

Trüg et al. [7] concluded that the best planners in that time were not yet able to solve real-world instances. They also observe that the core of the problem is to resolve the conflicts that arise when several airplanes have to access the same segments and that the automated planners are not aware of this fact.

As mentioned in the introduction, our interest in the airport stems mainly from its similarities with Sokoban and other transportation problems with a large number of goal states. Apart from airport-specific blocking rules, both problems share the main difficulties which come from the interaction of movable objects, and, for the variant of the airport ground traffic control problem with free goal positions, from the $k!$ goal states. More generally, a larger class of block-moving games shares these characteristics. In a previous work was identified the weakness of applying pattern databases to problems with lots of goal states, and a decomposition of the instance to overcome it was proposed [8], [9], [10]. Here we are interested to see if this weakness applies to the airport ground traffic control problem, and if it may be overcome with similar techniques.

As the airport domain is one of the problems introduced in the IPC 2004, we decided to use the solver Fast Downward, which has won the IPC 2004 [11]. Fast Downward already brings the IPC 2004 airport benchmark in the Planning Domain Definition Language (PDDL) format. It also has a large set of admissible domain-independent heuristics. These two characteristics make the results of Fast Downward easy to compare to our work.

III. A Domain-Dependent Heuristic for the Airport Domain

A domain-independent heuristic does not have prior knowledge about the structure of the problem. Consider, for example, a domain-independent pattern database heuristic. In the airport domain such heuristic would have to compute for each airplane the distance from every segment to every goal position by a reverse search. However, we know that the distances for all airplanes are the same, independent of other characteristics such as the airplane's size. Thus, we can compute the distances only once and reuse them for all airplanes. This is an example where we can use knowledge about the problem structure to improve the efficiency of the heuristic.

In the version of the airport domain with free goal positions an in-bound airplane can park at any parking segment. This is similar to Sokoban, where a box must be pushed to one of many goal squares. However, in the goal state each at goal square must be exactly one box. The standard approach to compute the heuristic function in Sokoban is to compute a minimum cost perfect matching between stones and goal squares, covering all goal squares with stones. Inspired by this approach, we propose a matching based heuristic for the Airport Domain.

Pre-Processing

We extract a directed graph for each instance whose vertices represent the segments of the airport. Two segments (u, v) are connected if there is an action that moves an airplane from a segment u to segment v. We then compute the shortest path between all pairs of segments of this graph and store the distances in a look-up table. This is done in a pre-processing phase, and all airplanes use the same distances during the search to compute the heuristic values. These distances are the shortest path between every pair of segments without taking into account path conflicts, blocked segments or any other interactions between the airplanes.

Closest Goal Heuristic

Our first approach is the domain-dependent closest goal heuristic. The closest goal heuristic is the sum of the distances required for all airplanes to reach their goal positions using the pre-computed distances. For the version with fixed goal positions, we look up the distance for each airplane to reach its specific goal position. In the version with free goal positions, we have to find the closest goal position for each airplane. For both versions, the cost for computing the heuristic is linear in the number of airplanes, since we can pre-compute the distances in constant time and then calculate the closest goal position for each airplane in each state. Additionally, each out-bound airplane must start up its engine and finally take off. Thus we increase the heuristic by one or two for each out-bound airplaine according to its current mode. Similarly, in-bound airplanes must shut down their engine to park, so we increase the heuristic by one for each in-bound airplane whose engine is still running.

In the version with fixed goal positions, with a_i in-bound airplanes the closest goal heuristic returns the cost to cover a_i parking positions. This is not the case in the version with free goal positions. In the latter version, all a_i airplanes can be mapped to the same parking position, which is not a valid solution. In this case, we could lose up to the sum of the $a_i - 1$ farthest parking positions in the heuristic value.

Matching Heuristic

Another natural heuristic to solve the airport domain problem is based on matchings. To compute the minimum distance for all airplanes to reach a different goal position, we construct

a bipartite graph. In this graph one part of the vertices represents the airplanes and the other part of the vertices represents all possible goal positions. Between each airplane and goal position, we add an edge with a weight corresponding to the length of the shortest path the airplane can take to reach that goal position. Since the number of goal positions exceeds the number of airplanes, we add a suitable number of dummy airplanes to make the cardinality of the two parts equal. These airplanes have distances of 0 to each goal position. Then a minimum weight perfect matching of airplanes and goal positions is a lower bound on the number of moves needed to bring each airplane to a different goal position.

We compute the minimum perfect matching using the Blossom algorithm [12] and also using the pre-processed distance information extracted from the instances. The matching can be seen as an extension of the closest goal approach. The difference is that while the closest goal heuristic tries to minimize the individual distance for each airplane to its goal position without taking into account multiple airplanes occupying to the same goal position, the matching heuristic tries to minimize the sum of the distances ensuring that two airplanes cannot be moved to a same goal position. As for the closest goal heuristic, we can increase the heuristic value by the number of outstanding engine shutdown operations.

The algorithm considers only airplanes that must be parked. Since the segments of out-bound airplanes will become available for other airplanes after take-off, every out-bound airplane could head for the same runway segment. We can notice that in the fixed goal positions variant of the problem the minimum perfect matching is trivial, since each vertex of the bipartite graph would have only one edge and would be equivalent to the closest goal approach.

IV. Experimental Results

In this section we report computational experiments comparing domain-independent and domain-dependent heuristics on the airport domain with fixed and free goal positions. For the experiments we use the stable version 1.4 of the Fast Downward planner [11]. We have chosen Fast Downward for our implementation to be able to compare better to the domain-dependent heuristics, and have implemented our domain-dependent heuristics within the framework provided by the planner.

In the following we describe the experiments on these instances. The first experiment evaluates six standard heuristics, on the variants with fixed as well as free goal positions, the second analyzes the closest goal heuristic on the variant with fixed goal positions, and the third analyzes the closest goal and the matching heuristic on the variant with free goal positions. All experiments have been run on a PC with an AMD FX-8150 processor running at 1.4 GHz and 32 GB of main memory. All domain-independent heuristics have been run with their default parameters. We have imposed a time limit of 30 minutes and a memory limit of 4 GB for each run.

TABLE I: Results for domain-independent heuristics.

	Fixed paths			Free paths		
Heur.	Opt.	Nodes	t	Opt.	Nodes	t
blind	22	1925645.7	8.3	23	2356855.9	10.7
$h_{\max}$	23	191425.6	115.8	23	417427.0	167.7
LM-cut	28	2508.2	44.7	26	19154.3	95.5
M&S	18	669808.3	127.4	18	910637.7	142.0
GAPDB	25	808624.5	7.1	27	1591448.8	14.9
iPDB	28	1594.0	258.7	28	68120.0	236.3

Evaluation of domain-independent heuristics

In our first experiment we evaluate standard domain-independent heuristics on the airport domain. All heuristics have been run with their default parameters. Table I shows the results for blind search, the heuristics $h_{\max}$ [13], landmark-cut [14] (LM-cut), merge-and-shrink [15] (M&S), and two heuristics based on pattern databases: a pattern database which uses a genetic algorithm to create the patterns ("GAPDB") [16], and the iPDB [17], [18]. For each heuristic we report the number of instances solved to optimality ("Opt."), and the average number of explored nodes ("Nodes") and average time ("t") in seconds over the optimally solved instances.

The standard domain-independent heuristics are able to solve between 18 and 28 instances. Heuristics LM-cut and iPDB perform best for fixed as well as free goal positions. We can see that all heuristics need more nodes and more time to solve the instances with free goal positions, as expected. However, this is not reflected in the number of solved instances. While LM-cut solves two instances less, iPDB solves the same number of instances, and blind search and the GAPDB heuristic even solve more. This happens mostly in instances with a high number of out-bound airplanes, since the relaxation of the take-off positions simplifies the problem.

The closest goal heuristic on instances with fixed goal positions

In our second experiment we have evaluated the closest goal heuristic on the instances with fixed goal positions. We compared the results for the two best standard heuristics LM-cut and iPDB, and the closest goal heuristic. For the instances which could not be solved by at least one method, we report in Table II the initial heuristic value ("h") and the best f-value ("f") – the lowest f-value on the open list when the time limit is reached or the instance is solved.

The closest goal heuristic solves 32 instances, four more than LM-cut and iPDB. The solution time of closest goal is always less (with the exception of instance 23) and often by an order of magnitude. Comparing only instances which were solved by all three methods, the closest goal heuristic needs only $61s$ while LM-cut needs $603s$ and iPDB $3619s$. However, LM-cut and iPDB expand significantly fewer nodes than the closest goal heuristic. The closest goal initial heuristic value is weaker than iPDB when the number of airplanes is relatively

TABLE II: Values of the initial heuristic (h) and best f-value (f) for instances with fixed parking positions. Best initial heuristic values or best f-values are highlighted. Only instances which were not solved by at least one method and present different values for initial h and best f-values are shown. Instances for iPDB that were not solved and present f-value equals to initial heuristic value reached the time limit before the conclusion of the pattern database construction.

	LM-cut		iPDB		Closest	
Inst.	h	f	h	f	h	f
31	358	358	357	357	358	358
32	390	390	387	387	390	390
33	393	393	388	388	393	393
34	427	427	422	422	427	427
35	433	433	425	425	433	433
39	208	208	210	210	208	210
40	190	190	191	191	190	191
41	178	178	179	179	178	179
42	257	257	259	259	257	257
43	223	223	224	224	223	223
44	227	227	229	229	227	227
45	249	249	251	251	249	249
46	290	290	292	292	290	290
48	423	423	399	399	423	423
49	451	451	447	447	451	451
50	685	685	670	670	685	685
Avg.	**301.9**	301.9	299.8	299.8	**301.9**	**302.1**

small compared to the number of goal segments, however, it does not necessarily impact the final f-value. Also, the initial h-value for closest goal and LM-cut get much better as the number of goal variables increases (larger instances). All the instances that were not solved reached the time limit for the three methods.

The closest goal and the matching heuristic on instances with free goal positions

In the third experiment we have evaluated the closest goal and the matching heuristic on the instances with free goal positions. Table III reports the number of nodes ("Nodes") and the time ("t") for LM-cut, iPDB, the closest goal and matching heuristics on the instances which were solved by at least one approach. In Table IV we show the initial heuristic value ("h") and the final f-value ("f") for instances which were not solved by at least one method.

With free goal positions LM-cut was able to solve 26, iPDB 28, closest goal 30 and matching 33 instances. As expected, we observe a degradation in performance of all methods, due to the larger number of goal states, and worse heuristic estimates. The performance degradation is limited, since the instances have a small number of airplanes. There are only 25 instances with more than two in-bound airplanes, and only 8 of them were solved with fixed goal positions, and the performance degradation is limited to the latter. This also explains why the matching heuristic is not substantially better than the closest goal heuristic. The difference will be only visible, for a larger

TABLE III: Comparison of the closest goal and matching heuristics to the two best domain-independent heuristics on the instances with free parking positions. Only instances which were solved by at least one approach are shown. Instances marked with † were also solved by blind search.

	LM-cut		iPDB		Closest		Matching	
Inst.	Nodes	t	Nodes	t	Nodes	t	Nodes	t
1†	9	0	9	0	9	0	9	0
2†	10	0	10	0	11	0	11	0
3†	20	0	24	0	34	0	34	0
4†	21	0	21	0	21	0	21	0
5†	22	0	22	0	23	0	23	0
6†	65	0	42	2	43	0	43	0
7†	65	0	43	2	45	0	45	0
8†	425	0	247	7	639	0	639	0
9†	793	2	2399	39	7559	0	6982	0
10†	19	0	19	0	19	0	19	0
11†	22	0	22	0	23	0	23	0
12†	40	0	40	2	41	0	41	0
13†	50	0	39	1	41	0	41	0
14†	61	0	40	2	65	0	65	0
15†	155	0	64	14	105	0	105	0
16†	402	1	94	123	307	0	307	0
17†	1766	11	321	436	4423	0	4337	0
18	8358	81	119075	1557	1102988	68	1101462	63
19†	419480	1194	763101	155	1722408	70	55108	3
20	-	>	-	>	-	>	569285	41
21†	96	1	96	10	97	0	97	0
22†	54311	596	54368	56	67148	6	146	0
23	-	>	966423	1513	1220965	170	150	0
24	158	7	157	585	217	0	217	0
25	-	>	-	>	-	>	137419	49
26	-	>	-	>	-	>	295878	211
27	1502	269	-	>	2073	1	2073	1
36†	79	1	79	22	79	0	79	0
37†	5435	185	154	224	22602	4	22602	12
38†	4648	1323	118	261	21075	3	21075	9
39	-	>	-	>	1644337	374	1644337	977
40	-	>	197	752	982983	277	982983	581
41	-	>	136	843	399723	99	399723	203
Tot.	26	15083	28	15616	30	6478	**33**	**2158**

numbers of in-bound airplanes. For example, in instance #39, which has no in-bound airplanes, both heuristics need the same number of nodes. In contrast, in instance #23, which has 3 in-bound airplanes, the matching heuristic is substantially better. However, it is remarkable that the matching heuristic dominates closest goal on the initial states and in instances where the number of in-bound airplanes is strictly greater than the number of out-bound airplanes the matching heuristic also leads to a significantly higher lower bound than closest goal. The matching heuristic also expands fewer nodes than the closest goal heuristic when the number of in-bound planes is equal or greater than the number of out-bound airplanes.

In 47 of the 50 instances the matching heuristic presents the best final f-value over all methods. In 17 of the 25 instances presented in Table IV the lower bound for matching

TABLE IV: Values of the initial heuristic (h) and best f-value (f) for instances with free parking positions. Best initial heuristic values or best f-values are highlighted. Only instances which were not solved by at least one method and present different values for initial h and best f-values are shown. Instances for iPDB that were not solved and present f-value equals to initial heuristic value reached the time limit before the conclusion of the pattern database construction.

	LM-cut		iPDB		Closest		Matching	
Inst.	h	f	h	f	h	f	h	f
20	113	113	113	113	113	113	115	115
23	144	144	144	144	144	148	148	148
25	206	206	206	206	206	206	208	208
26	196	196	196	196	196	196	200	200
28	288	288	288	288	288	288	290	290
29	284	284	284	284	284	284	288	288
31	352	352	351	351	352	352	354	354
32	366	366	363	363	366	366	370	370
33	387	387	384	384	387	387	389	389
34	403	403	399	399	403	403	407	407
35	383	383	378	378	383	383	391	391
39	142	142	144	144	142	144	142	144
40	159	159	160	160	159	160	159	160
41	134	134	135	135	134	135	134	135
42	185	185	187	187	185	185	185	185
43	179	179	180	180	179	179	179	179
44	174	174	176	176	174	175	176	176
45	190	190	192	192	190	190	194	194
46	205	205	207	207	205	205	205	205
47	264	264	251	251	264	264	268	268
48	325	325	266	266	325	325	325	325
49	354	354	276	276	354	354	358	358
50	533	533	400	400	533	533	537	537
Avg	260.4	260.4	248.9	248.9	260.4	260.7	**262.6**	**262.8**

is better than the best f-value for LM-cut and closest goal. The same occurs for 16 instances when compared to iPDB. As the problem variant with fixed goal positions the initial heuristic value of iPDB is better than any other method when the number of airplanes is small compared to the number of goal segments. When the number of goal variables increases to 6 or more, iPDB becomes the weakest method. The closest goal heuristic is much less informative than the matching heuristic, but it performs better than iPDB and LM-cut when comparing lower bounds and number of instances solved.

V. Conclusions and Future Work

We have proposed two domain-dependent heuristics for the airport domain and compared their performance to standard domain-independent heuristics in the Fast Downward planner. We found that even relatively simple domain-dependent heuristic such as closest goal or matching can improve the results of powerful domain-independent heuristics. It is also remarkable to notice that in instances where the number of in-bound airplanes is larger than the number of out-bound airplanes the results of the closest goal and matching heuristics are significantly better than those of iPDB or LM-cut. In these instances the matching heuristic also presents better results than the closest goal heuristic as measured by the number of expanded nodes and initial and final lower bounds.

In future work, it is possible to apply other strategies that were successful in Sokoban, such as pattern searches [19] and linear conflicts [4]. Finally, one could propose new instances increasing the number of in-bound planes this would present a more realistic version of the domain, since in real-world airports the number of in and out-bound airplanes should be approximately the same. In this more realistic version, given our current evidence, the heuristics iPDB and LM-cut would likely have a greater performance degradation while the matching heuristic would maintain the same performance.

References

[1] J. Hoffmann, S. Edelkamp, S. Thiébaux, R. Englert, F. dos Santos Liporace, and S. Trüg, "Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4," *J. Artif. Intell. Res.*, vol. 26, pp. 453–541, 2006.

[2] W. Hatzack and B. Nebel, "The operational traffic control problem: Computational complexity and solutions," in *Rec. Adv. AI Planning, 6h Europ. Conf. Planning*, 2001.

[3] M. Helmert, "New complexity results for classical planning benchmarks," in *Proc. 16th Int. Conf. Autom. Plan. Sched.*, 2006, pp. 52–61.

[4] A. Junghanns and J. Schaeffer, "Sokoban: Enhancing general single-agent search methods using domain knowledge," *Artificial Intelligence*, vol. 129, no. 1–2, pp. 219–251, 2001.

[5] J. C. Culberson and J. Schaeffer, "Pattern databases," *Computational Intelligence*, vol. 14, no. 3, pp. 318–334, 1998.

[6] S. Edelkamp, "Planning with pattern databases," in *European Conference on Planning*, 2001, pp. 13–24.

[7] S. Trüg, J. Hoffmann, and B. Nebel, "Applying automatic planning systems to airport ground-traffic control – a feasibility study," in *Proc. 27th Annual German Conf. AI*, ser. LNCS, no. 3238, 2004, pp. 183–197.

[8] A. G. Pereira, M. Ritt, and L. S. Buriol, "Finding optimal solutions to Sokoban using instance dependent pattern databases," in *Proc. 6th Int. Symp. Comb. Search*, M. Helmert and G. Röger, Eds. AAAI Press, Sep. 2013.

[9] ——, "Solving Sokoban optimally using pattern databases for deadlock detection," in *Anais do XI Encontro Nacional de Inteligência Artificial (ENIA)*, 2014.

[10] ——, "Optimal Sokoban solving using pattern databases with specific domain knowledge," *Artif. Intell.*, vol. 227, pp. 52–70, 2015.

[11] M. Helmert, "The fast downward planning system." *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, 2006.

[12] J. Edmonds, "Paths, trees, and flowers," *Canad. J. Math*, vol. 17, pp. 449–467, 1965.

[13] B. Bonet and H. Geffner, "Planning as heuristic search," *Artif. Intell.*, vol. 129, pp. 5–33, 2001.

[14] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?" in *Proc. 19th Int. Conf. Autom. Plan. Sched.*, A. Gerevini, A. Howe, A. Cesta, and I. Refanidis, Eds., 2009, pp. 162–169.

[15] M. Helmert, P. Haslum, and J. Hoffmann, "Flexible abstraction heuristics for optimal sequential planning," in *Proc. 17th Int. Conf. Autom. Plan. Sched.*, 2007, pp. 176–183.

[16] S. Edelkamp, "Automated Creation of Pattern Database Search Heuristics," in *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence*, 2007, pp. 35–50.

[17] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig, "Domain-independent construction of pattern database heuristics for cost-optimal planning," in *Proc. 22th AAAI Conf. Art. Intel.*, 2007, pp. 1007–1012.

[18] S. Sievers, M. Ortlieb, and M. Helmert, "Efficient implementation of pattern database heuristics for classical planning," in *Proc. 5th Int. Symp. Comb. Search*, D. Borrajo, A. Felner, R. Korf, M. L. abd Carlos Linares Lopez, W. Ruml, and N. Sturtevant, Eds. AAAI Press, 2012, pp. 105–111.

[19] A. Junghanns and J. Schaeffer, "Single-agent search in the presence of deadlocks," in *AAAI/IAAI*, 1998, pp. 419–425.