Flow-based Heuristics for Optimal Planning: Landmarks and Merges

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Menkes van den Briel
NICTA & Australian National University
Canberra, Australia
menkes@nicta.com.au

Abstract

We describe a flow-based heuristic for optimal planning that exploits landmarks and merges. The heuristic solves a linear programming (LP) problem that represents variables in SAS^+ planning as a set of interacting network flow problems. The solution to the LP provides a reasonable admissible heuristic for optimal planning, but we improve it considerably by adding constraints derived from action landmarks and from variable merges. Merged variables, however, can quickly grow in size and as a result introduce many new variables and constraints into the LP. In order to control the size of the LP we introduce the concept of dynamic merging that allows us to partially and incrementally merge variables, thus avoiding the formation of cross products of domains as done when merging variables in the traditional way. The two types of improvements (action landmarks and variable merges) to the LP formulation are orthogonal and general. We measure the impact on performance for optimal planning of each improvement in isolation, and also when combined, for a simple merge strategy. The results show that the new heuristic is competitive with the current state of the art.

Introduction

We present three general domain-independent techniques for improving flow-based heuristics for optimal planning. Flow-based heuristics are heuristics defined by the solution of a linear program (LP) that is solved for each state encountered during the search and that tracks the presence of fluents throughout the application of actions in potential plans (van den Briel et al. 2007; Bonet 2013). Flow-based heuristics can be quite informative on some tasks as they are not bounded by the optimal delete-relaxation heuristic h^+. As a result, they offer a broad open ground for development and improvement. We partially explore this ground and show how simple and meaningful ideas result in substantially improved heuristics.

In our flow-based heuristic, we view planning as a set of interacting network flow problems. Each variable of the problem corresponds to a separate network flow problem where nodes correspond to the values of the variable and arcs correspond to transitions between these values. Interaction between these networks is because actions can impact multiple variables. The flow-based heuristic counts occurrences of actions, yet ignores the ordering between them. There are other LP-based heuristics for optimal planning (Katz and Domshlak 2010; Pommerening, Röger, and Helmert 2013) but these are not flow-based heuristics. On the other hand, Pommerening et al. (2014) present an unifying framework for LP-based heuristics for optimal planning.

An LP formulation based on flow relations is described by van den Briel et al. (2007). This work introduces a flow-based heuristic, but does not incorporate it into a planning system. The heuristic value at the initial state of several benchmarks problems is reported. One important observation is that by merging variables the heuristic value can be improved significantly, often achieving a heuristic value better than h^+. Bonet (2013), on the other hand, describes a very similar LP formulation based on flow relations that is derived from the state equation in Petri-nets. The LP formulation is incorporated in an A* search algorithm and performs very well in certain benchmark domains, but this LP formulation does not exploit merging.

In this paper we build upon these two flow-based heuristics by (1) automatically reformulating the planning problem in order to strengthen the solution of the LP, (2) adding constraints derived from information contained in action landmarks, and (3) by incorporating new variables and constraints obtained from merging variables. Our theoretical results show that the improved heuristics are all admissible and dominate well-known heuristics, while the empirical results show that the new heuristics are competitive with the state of the art for optimal planning.

It is important to note that landmarks are state dependent (landmarks that apply to one state may not necessarily apply to another state). As a result, landmark constraints are added and removed from the LP formulation each time we evaluate a state. Constraints and variables obtained from merging variables, on the other hand, apply to all states and therefore are never retracted from the LP once added.

Whenever we create a new merged variable, we represent it as a network flow problem just like all the other variables. Since merged variables typically have many more values than the variables it is composed of, representing the complete network flow problem of each merged variable can become quite costly. In this work we introduce the concept of *dynamic merging*. Dynamic merging is dynamic in all ways, meaning that we incrementally merge more and more vari-

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ables, but also that we incrementally merge the variables. Hence, throughout the process of dynamic merging, our LP formulation may only make explicit a small fraction of the network flow problem of a merged variable.

Dynamic merging allows us to be very selective with merging and prevents us from having to create the cross product of domains as done when merging variables in the traditional way. Specifically, in this work we only merge pairs of atoms (values), where one atom is the prevail condition and the other atom is the effect precondition of an action. The reason we target these pairs of atoms is that the corresponding network flow constraint is likely going to increase the solution value of the LP formulation, and therefore improve the estimate of the flow-based based heuristic. The selection of which atoms to merge is performed by a simple but effective merge strategy that analyzes and refines the LP formulation until reaching a fix point.

After presenting the framework for planning and the base formulation of the heuristic, we describe the incorporation of constraints derived from action landmarks and from variable merges. Results are presented throughout the paper to observe the impact of each improvement, while results for a final heuristic are presented at the end. The paper finishes with conclusions and future work.

Planning Problems

A SAS$^+$ planning problem with action costs is a tuple $P = \langle V, A, s_\circ, s_\star, c\rangle$ where V is a set of variables, each variable $X \in V$ with finite domain D_X, A is a set of actions, $s_\circ$ is the initial state, $s_\star$ is a goal description, and $c : A \to \mathbb{N}$ are non-negative action costs.

A valuation is an assignment of values to variables in V. When all variables are assigned a value it is a complete valuation, otherwise it is a partial valuation. The state model for P is made of the states corresponding to all the different complete valuations for the variables in V, with the initial state given by $s_\circ$ and the goal states given by all the states that agree with the partial valuation $s_\star$. For valuation s (either complete or partial) and variable X, we denote the value of X at s by $s[X]$. An atom for the problem P is a literal of the form $X = x$ where X is a variable and $x \in D_X$; we often abuse notation and say that such an atom p belongs to s when $s[X] = x$. For atom p, Var(p) and Val(p) denote the variable and value in p respectively.

An action in SAS$^+$ is a triple $\langle$Pre, Post, Prev$\rangle$ where the precondition Pre, the postcondition Post and the prevail condition Prev are all partial valuations, with the restrictions that any variable X that gets a value in Pre must also get a value in Post, and any variable X that gets a value in Prev must not get a value in either Pre or Post. We assume the standard interpretation of actions that defines applicability and the transition function $res : A \times S \to S$ mapping applicable actions and states into resulting states. If Pre and Post are defined on a common variable X, we assume Pre$[X] \neq$ Post$[X]$ since otherwise the condition on X should appear as a prevail and not as an effect of the action. Typically, if Post is defined on X then Pre is also defined, but this is not required. We say that an action a is *incomplete* for X when $X \in$ Var(Post) but $X \notin$ Var(Pre), and that a is incomplete when it is incomplete for at least one variable X.

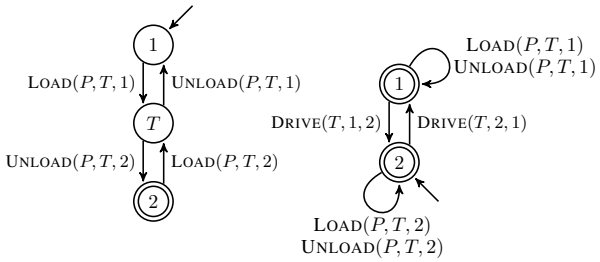

Figure 1: Domain transition graphs for the two variables $pos(package)$ (left) and $pos(truck)$ (right) in a simple logistics problem. In each DTG, the nodes represent the values of the variable, the arcs represent the transitions, and the loop arcs represent action prevail conditions. The initial state is indicated with a diagonal arrow and goal states are marked with an extra circle.

A plan is a sequence π of actions that defines a path from $s_\circ$ to a goal state. Its cost is the sum of the costs for the actions in π, and it is optimal if it is a minimum-cost plan.

Domain Transition Graphs

Let $P = \langle V, A, s_\circ, s_\star, c\rangle$ be a planning problem. The domain transition graph (DTG) for a variable $X \in V$ is a labeled directed graph with nodes for each value in D_X and a labeled arc for each transition in X, where each label corresponds to an action in the planning problem. Note, it is possible for an arc to have multiple labels.

Formally, the DTG for variable X is the tuple $\mathcal{A}_X = \langle D_X^*, L_X, T_X\rangle$ where $D_X^* = D_X \cup \{\bot\}$, $L_X = \{a \in A : X \in$ Var(Prev) $\cup$ Var(Pre) $\cup$ Var(Post)$\}$, and $T_X \subseteq D_X^* \times L_X \times D_X^*$ is the set of transitions (x, a, x') given by:

$$\begin{aligned} T_X \;=\; & \{(x,a,x') : X{=}x \in \text{Pre},\ X{=}x' \in \text{Post}\} \;\cup \\ & \{(\bot,a,x) : X \notin \text{Var(Pre)},\ X{=}x \in \text{Post}\} \;\cup \\ & \{(x,a,x) : X{=}x \in \text{Prev}\}\,. \end{aligned}$$

The undefined value $\bot$ is added as a state of $\mathcal{A}_X$ to account for the incomplete actions for X. If no such action exist then there is no need to consider the state $\bot$.

Figure 1 shows the DTGs for a simple logistics problem with one truck, one package, and two locations. Initially the package is at location 1, the truck is at location 2, and the goal is to have the package at location 2. This problem can be formulated with the variable $pos(truck) \in \{1, 2\}$ to indicate the position of the truck, and the variable $pos(package) \in \{1, 2, T\}$ to indicate the position of the package, which is either at location 1 or 2, or the truck (T).

Let X and Y be two variables with domains D_X and D_Y, and DTGs $\mathcal{A}_X = \langle D_X^*, L_X, T_X\rangle$ and $\mathcal{A}_Y = \langle D_Y^*, L_Y, T_Y\rangle$ respectively. The merging of variables X and Y gives the variable $Z = XY$ with domain $D_Z = \{(x, y) : x \in D_X, y \in D_Y\}$. The DTG $\mathcal{A}_Z = \langle D_Z^*, L_Z, T_Z\rangle$ for Z is the *parallel composition* of $\mathcal{A}_X$ and $\mathcal{A}_Y$ (cf. Dräger, Finkbeiner, and Podelski (2006)): $D_Z^* = D_X^* \times D_Y^*$, $L_Z = L_X \cup L_Y$,

and $((x, y), a, (x', y')) \in T_Z$ iff

$$(x, a, x') \in T_X \ \wedge \ (y, a, y') \in T_Y, \text{ or} \quad (1)$$

$$(x, a, x') \in T_X \ \wedge \ y = y' \ \wedge \ a \notin L_Y, \text{ or} \quad (2)$$

$$x = x' \ \wedge \ a \notin L_X \ \wedge \ (y, a, y') \in T_Y \, . \quad (3)$$

In some cases, the DTG $\mathcal{A}_Z$ may contain transitions that are impossible to occur as they correspond to valuations that are mutex[1] with the precondition or postcondition of the action. Hence, the parallel composition can be reduced by pruning some of such transitions in a manner that resembles the construction of constrained PDBs (Haslum, Bonet, and Geffner 2005). Indeed, the transitions given by case (1) are required to satisfy $X = x, Y = y \not\perp$Pre, those given by case (2) to satisfy $Y = y \not\perp$Pre and $Y = y \not\perp$Post, and those given by case (3) to satisfy $X = x \not\perp$Pre and $X = x \not\perp$Post, where $\nu \not\perp \mu$ means that the valuations ν and μ are non-mutex.

Flows and Base Model

A flow for a planning task $P = \langle V, A, s_\circ, s_\star, c \rangle$ is a function $f : A \to \mathbb{R}^+$ mapping action labels into non-negative real numbers.

For a fixed atom p and action a, we identify 4 possible behaviours of a on p: (1) p is *produced* by a if $p \in$ Post, (2) p is *consumed* by a if $p \in$ Pre, (3) p is *preserved* by a if $p \in$ Prev, and (4) p is *irrelevant* to a in other cases. These behaviours are exhaustive and mutually exclusive for each atom p and action a. Also an action a may produce an atom p without consuming an atom q on the same variable Var(p), but this is only possible when a is incomplete for Var(p).

A plan π defines an integral flow f_π where $f_\pi(a)$ is the number of occurrences of a in π. By analyzing the possible action behaviours on a fixed atom p, it is not difficult to obtain the following *flow balance equation* (van den Briel et al. 2007; Bonet 2013):

$$\#\text{ times } p \text{ is produced} \ - \ \#\text{ times } p \text{ is consumed} \ \geq \ \Delta_p$$

where Δ_p is the "net difference" for p between the goal state $s_\star$ and the initial state $s_\circ$; i.e.,

$$\Delta_p = \begin{cases} +1 & \text{if } s_\circ \not\models p \text{ and } s_\star \models p \, , \\ -1 & \text{if } s_\circ \models p \text{ and } s_\star \not\models p \, , \\ 0 & \text{otherwise} \, . \end{cases}$$

The first statement in the definition of Δ_p tells that p must be produced more times than it is consumed as p must hold at the goal state but does not hold at the initial state. The second tells that p may be consumed at most one more time than it is produced as it holds in the initial state but it is not required to hold at the goal state. The final statement covers the cases when $s_\circ$ and $s_\star$ both the same value for p as in both cases the required net difference is zero. The constraint in the flow balance equation is an inequality and not an equality because the presence of incomplete actions. If no such actions exist, the inequality can be "promoted" to an equality.

[1] Atoms p and q are *mutex* if no reachable state makes both true. Two partial assignments ν and μ are mutex if there are atoms $p \in \nu$ and $q \in \mu$ that are mutex.

Var(p) is a goal variable

	p isn't mutex with $s_\star$		p is mutex with $s_\star$	
	$s_\circ \models p$	$s_\circ \not\models p$	$s_\circ \models p$	$s_\circ \not\models p$
LB_p	0	1	-1	0
UB_p	∞ / 0	∞ / 1	∞ / -1	∞ / 0

Var(p) is not a goal variable

	p isn't mutex with $s_\star$		p is mutex with $s_\star$	
	$s_\circ \models p$	$s_\circ \not\models p$	$s_\circ \models p$	$s_\circ \not\models p$
LB_p	-1	0	-1	0
UB_p	∞ / 0	∞ / 1	∞ / -1	∞ / 0

Table 1: Different cases defining the lower and upper bound values on the flow balance equation for atom p.

The above flow balance equation for atom $p = X = x$ is formally cast as the equation over flows $f : A \to \mathbb{R}^+$:

$$LB_p \ \leq \sum_{(x', a, x) \in T_X} f(a) \ - \sum_{(x, a, x') \in T_X} f(a) \ \leq \ UB_p \quad (4)$$

where T_X is the set of transitions for the DTG $\mathcal{A}_X$ for variable X, and LB_p and UB_p are lower and upper bounds for the flow balance on p. Naively, $LB_p = \Delta_p$ and $UB_p = \infty$ but tighter bounds can be obtained by a detailed account on the relation between p, the goal and the actions. Indeed, we consider two cases whether Var(p) is mentioned in the goal or not (i.e., it is a goal variable), and subcases on whether the initial state satisfies p and the goal is mutex with p. For each case, Table 1 gives lower and upper bound values on the flow balance equation for atom p. The table has two upper bound values for each case. Here we refer to the first values; e.g., the lower and upper bounds for the equation for atom p when Var(p) is a goal variable, $s_\circ \models p$ and $s_\star$ is mutex with p is $(-1, \infty)$. It is easy to check that the values in Table 1 subsume Δ_p.

The first upper bound values are all equal to ∞, and thus impose no constraint on the flow, due to the presence of incomplete actions that may produce an atom that already holds in a given state. Under certain conditions the upper bounds can be tightened. One such condition is when the atom p refers to a variable X for which every action that produces an atom $X = x$, consumes another atom $X = x'$ for $x \neq x'$ and $x, x' \in D_X$. In such a case, the second values in the table for the upper bound can be used in place of the first values. As a result, when X is a goal variable or p is mutex with the goal state, the constraints on the flow balance equation for p become tight and collapse into an equality. This is the promotion of inequalities to equalities described by Bonet under the name of safeness-based improvement.

The bottom part of Table 1 shows that for non-goal variables Var(p) the bounds when p is non-mutex with $s_\star$ are not as tight as when p is mutex with $s_\star$. Thus, it makes sense to analyze the input task P to deduce as much as possible the atoms that are mutex with $s_\star$. We perform this computation in polynomial time by marking an atom p as mutex with $s_\star$ if either Var(p) is a goal variable and $s_\star$ assigns a different value to it, or there is an atom q mutex with p with $s_\star \models q$.

Domain	LM-cut	base
airport (50)	**28**	20
barman-opt11-strips (20)	**4**	**4**
blocks (35)	**28**	**28**
depot (22)	**7**	**7**
driverlog (20)	**13**	11
elevators-opt08-strips (30)	**22**	9
elevators-opt11-strips (20)	**17**	7
floortile-opt11-strips (20)	**6**	4
freecell (80)	15	**35**
grid (5)	**2**	1
gripper (20)	**7**	6
logistics00 (28)	**20**	15
logistics98 (35)	**6**	2
miconic (150)	**141**	50
mprime (35)	**22**	18
mystery (30)	**19**	15
nomystery-opt11-strips (20)	**14**	10
openstacks-opt08-strips (30)	**20**	15
openstacks-opt11-strips (20)	**15**	7
openstacks-strips (30)	**7**	**7**
parcprinter-08-strips (30)	18	**28**
parcprinter-opt11-strips (20)	13	**20**
parking-opt11-strips (20)	**2**	1
pathways-noneg (30)	**5**	4
pegsol-08-strips (30)	27	**28**
pegsol-opt11-strips (20)	17	**18**
pipesworld-notankage (50)	**17**	15
pipesworld-tankage (50)	**11**	10
psr-small (50)	49	**50**
rovers (40)	**7**	6
satellite (36)	**7**	6
scanalyzer-08-strips (30)	**15**	13
scanalyzer-opt11-strips (20)	**12**	10
sokoban-opt08-strips (30)	**28**	16
sokoban-opt11-strips (20)	**20**	15
tidybot-opt11-strips (20)	**13**	5
tpp (30)	6	**8**
transport-opt08-strips (30)	**11**	10
transport-opt11-strips (20)	**6**	**6**
trucks-strips (30)	**10**	9
visitall-opt11-strips (20)	10	**17**
woodworking-opt08-strips (30)	**16**	12
woodworking-opt11-strips (20)	**11**	7
zenotravel (20)	**12**	9
Total (1396)	**756**	594

Table 2: Coverage results for optimal planning for the LM-cut and base heuristics. Best results are highlighted.

Further, whenever there is a variable X such that all atoms $X = x'$, for $x' \in D_X \setminus \{x\}$, are marked as mutex with $s_\star$, then X is marked as a goal variable and the goal is extended with $X = x$. This computation is performed until reaching a fix point as a preprocessing step.

The base LP model for the problem $P = \langle V, A, s_\circ, s_\star \rangle$ is defined on the problem P' that results of the preprocessing performed on P. The LP has $|A|$ variables, denoted by $f(a)$ for each action $a \in A$, that define a flow for P. The objective function is to minimize $\sum_{a \in A} c(a) \times f(a)$, while the constraints are the flow balance equations for all atoms $X = x$ for each variable $X \in V$ and value $x \in D_X$, and the non-negative constraints $f(a) \geq 0$ for each $a \in A$. Except for the preprocessing, this LP is the same as the one used by Bonet (2013) to define the SEQ heuristic *enhanced* with 1-safe information, while the plain SEQ heuristic corresponds to this LP but with all upper bounds set to ∞ as given by the first values shown in Table 1. In both cases the solution to the LP provides an admissible estimate for the optimal cost and P has no solution if the base model is infeasible. The base LP defines a heuristic h_{base} whose value at state s is the cost of the solution of the LP for problem $P[s_\circ = s]$,[2] where $P[s_\circ = s]$ refers to the problem P but with the initial state set to s. The intuition behind the following result is that the base LP only considers the flow balance equation (4) for each atom p which must be satisfied by any feasible plan.

Theorem 1. *The heuristic h_{base} is admissible. This holds whether the first or second upper bounds in Table 1*

For illustration, let us consider the gripper domain that consists of a two-armed robot that must transport a number of balls from room A to room B. The robot can move from one room to another, and pick/drop balls into/from either room and from either gripper. To pick a ball, the robot must be at the ball's room and the gripper must be holding nothing. An instance of gripper with n balls can be formulated with the variable $pos(robby) \in \{A, B\}$ that tells the position of the robot, and variables $pos(b_i) \in \{L, R, A, B\}$, for $i = 1, \ldots, n$, that tell the position of the balls, either left (L) or right (R) gripper, or the room A or B. An action like PICKLEFT(b_i, A) has prevail $pos(robby) = A$, precondition $pos(b_i) = A$, postcondition $pos(b_i) = L$, and all other atoms as irrelevant. The flow balance equation for the atom $p =$ '$pos(b_i) = B$' is

$$f(\text{DROPLEFT}(b_i,B)) + f(\text{DROPRIGHT}(b_i,B)) - f(\text{PICKLEFT}(b_i,B)) - f(\text{PICKRIGHT}(b_i,B)) = 1$$

since DROPLEFT(b_i, B) and DROPRIGHT$(b_i, B)\}$ are the actions that "produce" p, PICKLEFT(b_i, B) and PICKRIGHT$(b_i, B)\}$ are the actions that "consume" p, and ball b_i is initially at room A and at room B in the goal.

In a first experiment we compare the effectiveness of h_{base} with respect to LM-cut (Helmert and Domshlak 2009), the current state of the art for optimal planning, over a comprehensive benchmark of problems. Table 2 shows coverage results on each domain.[3] As can be seen, overall, LM-cut amply dominates h_{base}, yet the latter shows promising results on some domains; specifically, on freecell, parcprinter, tpp, and visitall. This and the following experiments are performed with LPs that only contains lower bound constraints (i.e., the UB_p values are set to infinity). As shown by Pommerening et al. (2014), these constraints imply the upper bound values in Table 1 (we come back to this issue below).

In the following, we explore different ways to improve the estimates offered by the base model.

Landmarks

We first study the improvement suggested by Bonet (2013) about adding constraints for the action landmarks of the problem. An action landmark L, or just landmark, for a task $P = \langle V, A, s_\circ, s_\star, c \rangle$ is a subset of actions such that every plan for P contains at least one action in L (Hoffmann, Porteous, and Sebastia 2004; Helmert and Domshlak 2009). Such a landmark implies the lower bound $c(L)$ on the optimal cost h^* where $c(L) = \min_{a \in L} c(a)$. Given a collection $\mathcal{L} = \{L_i\}_{i=1}^n$ of landmarks, one can obtain an estimate on h^* in at least three different ways: (1) by computing the maximum $c(L_i)$ over $L_i \in \mathcal{L}$, (2) by computing a cost partitioning $\mathcal{C} = \{c_i\}_{i=1}^n$ for $\mathcal{L}$ and then forming the sum $h_{\mathcal{C}} = \sum_{i=1}^n c_i(L_i)$ (Karpas and Domshlak 2009), or (3) by solving the hitting set problem for $\mathcal{L}$ (Bonet and Helmert 2010). These three estimates are admissible, give values that are monotonically non-decreasing, but at increasing computation costs.

A cost partitioning for $\mathcal{L} = \{L_i\}_{i=1}^n$ is a collection $\mathcal{C} = \{c_i\}_{i=1}^n$ of non-negative cost functions such that $\sum_{i=1}^n c_i(a) \leq c(a)$ for every $a \in A$. An optimal cost partitioning $\mathcal{C}^*$ for $\mathcal{L}$ is such that $h_{\mathcal{C}^*} \geq h_{\mathcal{C}}$ for every other cost partitioning $\mathcal{C}$ for $\mathcal{L}$. An optimal cost partitioning and

[2] If all operator costs are integral, the value of the solution for the LP can be rounded up to the next integer without loosing admissibility.

[3] All experiments were run with Fast Downward (Helmert 2006) on AMD Opteron 6378 CPUs running at 2.4 GHz with time and memory limits of 1,800 seconds and 2Gb respectively. The LP solver is IBM CPLEX v.12.5.1.

Domain	base	$f10$	$f20$
airport (50)	20	**26**	25
barman-opt11-strips (20)	**4**	**4**	**4**
blocks (35)	28	28	**29**
depot (22)	7	7	7
driverlog (20)	11	11	**13**
elevators-opt08-strips (30)	9	9	**18**
elevators-opt11-strips (20)	7	7	**15**
floortile-opt11-strips (20)	4	4	**6**
freecell (80)	35	**47**	27
grid (5)	1	**2**	1
gripper (20)	6	**7**	5
logistics00 (28)	15	14	**20**
logistics98 (35)	2	3	**6**
miconic (150)	50	57	**140**
mprime (35)	18	**20**	**20**
mystery (30)	15	**16**	**16**
nomystery-opt11-strips (20)	10	8	**12**
openstacks-opt08-strips (30)	**15**	12	**15**
openstacks-opt11-strips (20)	7	7	7
openstacks-strips (30)	7	**8**	7
parcprinter-08-strips (30)	28	28	**29**
parcprinter-opt11-strips (20)	**20**	**20**	**20**
parking-opt11-strips (20)	**1**	**1**	**1**
pathways-noneg (30)	4	4	**5**
pegsol-08-strips (30)	**28**	26	27
pegsol-opt11-strips (20)	**18**	16	17
pipesworld-notankage (50)	15	**16**	11
pipesworld-tankage (50)	**10**	**10**	9
psr-small (50)	**50**	**50**	**50**
rovers (40)	6	6	**7**
satellite (36)	6	6	**7**
scanalyzer-08-strips (30)	**13**	**13**	11
scanalyzer-opt11-strips (20)	**10**	9	8
sokoban-opt08-strips (30)	16	20	**27**
sokoban-opt11-strips (20)	15	16	**20**
tidybot-opt11-strips (20)	5	**11**	8
tpp (30)	**8**	**8**	**8**
transport-opt08-strips (30)	10	10	**11**
transport-opt11-strips (20)	**6**	5	**6**
trucks-strips (30)	**9**	**9**	**9**
visitall-opt11-strips (20)	17	17	**19**
woodworking-opt08-strips (30)	12	14	**20**
woodworking-opt11-strips (20)	7	9	**15**
zenotravel (20)	9	9	**11**
Total (1396)	594	630	**749**

Table 3: Coverage results for optimal planning for the base heuristic extended with constraints from landmarks. The column $f10$ refers landmarks computed using the Weak-Zhu-Givan method, while $f20$ to landmarks computed using the LM-cut method. Best results are highlighted.

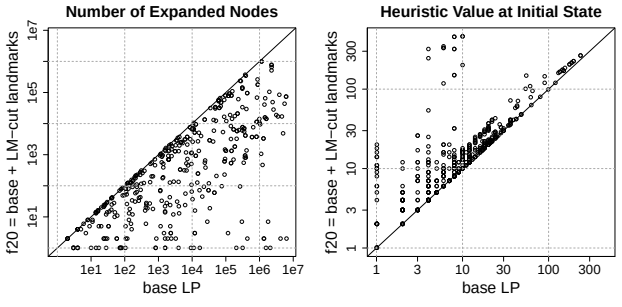

Figure 2: Number of expanded nodes and value at initial state for the base LP and $f20$ across common solved tasks.

its value can be computed in polynomial time by solving a simple LP (Karpas and Domshlak 2009).

The heuristic LM-cut can be understood as a method to compute a collection $\mathcal{L}_{\text{LMC}}$ of landmarks and a cost partitioning $\mathcal{C}_{\text{LMC}}$. LM-cut's value at the initial state is $h_{\mathcal{C}_{\text{LMC}}}$ while for other states s, it is the value for the task $P[s_\circ = s]$. We denote with $\mathcal{L}_{\text{LMC}}[s]$ the collection of landmarks computed by LM-cut for state s.

There are other algorithms for computing landmarks (Keyder, Richter, and Helmert 2010; Zhu and Givan 2003). In practice, however, it is relative expensive to compute a fresh set of landmarks for every state encountered during the search. One way to speed up such computation, at the cost of missing some landmarks, consists in first obtaining a collection $\mathcal{L}$ for the initial state, and then keeping track of the landmarks that get satisfied along the paths from $s_\circ$ to the reachable states s, with the aim of obtaining a collection for such a state s equal to the initial collection minus the landmarks that get satisfied along the paths leading to s. This is the method employed by the LA heuristic (Karpas and Domshlak 2009). For a method A for computing landmarks, we refer to this speed up by Weak-A.

Let us now focus on how to use a collection of landmarks $\mathcal{L}$ to improve the values given by the base LP. A landmark L for state s says that any plan for s must contain at least one action in L. Thus, L directly induces a constraint of the form $\sum_{a \in L} f(a) \geq 1$ on any flow f for P. Since the collection of landmarks depends on the state, the induced constraints cannot be permanently added to the LP. Rather, the constraints are added before solving the LP for the state and retracted afterwards. If A is a method for computing landmarks, we denote with $\mathcal{L}_A[s]$ the collection computed by A on state s, and with h^A_{base} the heuristic that results of solving the base LP extended with the set $Constr(\mathcal{L}_A[s])$ of induced constraints. From now on, we say that a constraint (or set of constraints) is admissible if the flow f_π of any plan π satisfies the constraint (or constraints in the set). We have

Theorem 2. *The set $Constr(\mathcal{L}_A[s])$ is admissible for $P[s_\circ = s]$ and thus h^A_{base} is an admissible heuristic. Further, h^A_{base} dominates h^*_A where h^*_A is the heuristic that assigns to state s the value of the optimal cost partitioning for $\mathcal{L}_A[s]$. In particular, $h^{\text{LM-cut}}_{base} \geq h^*_{\text{LM-cut}} \geq h_{\text{LM-cut}}$.*

Table 3 shows the overall performance of A* when using the base LP and the base LP extended with constraints for landmarks computed with the Weak-Zhu-Givan method (column $f10$) and the LM-cut method (column $f20$). As can be seen, there is a noticeable jump in the number of solved problems in both cases: for $f10$ the jump is 36 from 594 to 630, while for $f20$ the jump is 155 from 594 to 749. The coverage of a stronger heuristic, however, does not necessarily dominate the weaker heuristic. The experiments are performed with a time cutoff and the stronger heuristic usually requires more time; e.g., for scanalyzer-08, the base LP solves 13 tasks while $f20$ solves 11. Hence, the big jump in number of problems solved is due to better heuristic estimates for states. This is clearly seen in the scatter plots in Figure 2 for the number of expanded nodes[4] and the value at the initial state for the tasks solved by both heuristics.

Merges

The idea of merging variables for obtaining better heuristics may have originated from planning with pattern databases (Edelkamp 2001). The idea got more traction when, in the same year, Helmert, Haslum, and Hoffmann (2007) introduced the merge-and-shrink heuristic and van den Briel et al. (2007) introduced a flow-based heuristic similar to the one described in this paper, that exploits merging. While both works apply merging, they perform it in different ways.

In merge-and-shrink, variables are literally merged away. For example, after merging variables X and Y into the variable XY, only one variable exists, which is variable XY. In the heuristic by van den Briel et al. no variable is ever removed. Hence, after merging variables X and Y into XY, three variables remain, X, Y, and XY. The reason for not removing variables X and Y is that they can still be used in other merges. For example, we may only be interested in the

[4]Total number of expansions before last f-layer. Points with zero value omitted since plot is in log scale.

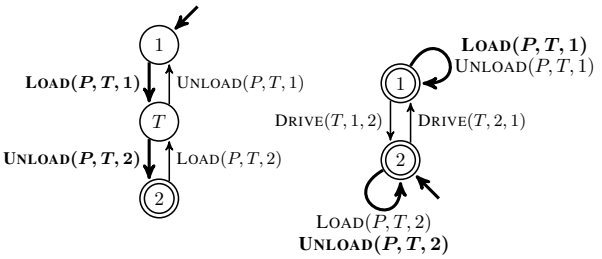

Figure 3: The solution of the base LP shown on top of the DTGs for the simple logistics example using highlighted arcs and action labels.

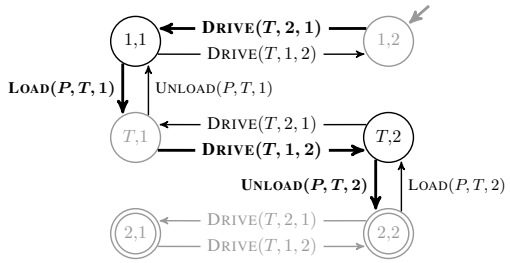

Figure 4: Single state variable obtained from merging the variables in the example. Only the highlighted nodes $merge(1,1)$ and $merge(T,2)$ are added to the LP formulation using dynamic merging. The solution to the LP after adding the two merges is shown using highlighted arcs and action labels, and it yields the perfect value of 4.

merged variables XY and XZ, yet if we had to merge variables away, we would be forced to create the merged variable XYZ whose size may be much bigger than the combined sizes of XY and XZ. In terms of our LP formulation, constraints corresponding to XY and XZ can both improve the heuristic value, and thus it is beneficial to consider both. Constraints corresponding to XYZ also improve the heuristic value, but this requires many more constraints in the LP.

In our flow-based heuristic we do not remove any variables after merging, but unlike van den Briel et al.'s approach we allow for *dynamic merging*, which enables us to only partially merge variables in order to manage the number of variables and constraints that are added to the LP. Dynamic merging is probably best explained by focussing on the atoms rather than on the variables. With dynamic merging we are merging only selected pairs of atoms p and q, where $\text{Var}(p) \neq \text{Var}(q)$, rather than creating the cross product of variable domains.

Dynamic merging has some overlap with the idea of creating conjunctions of atoms to boost both admissible and non-admissible heuristics (Haslum 2012; Keyder, Hoffmann, and Haslum 2012). These works, however, are syntactic as they translate a given task P into another task P' that has additional atoms representing the merged values. Our approach is different as we perform an efficient (polytime) merging that creates new variables and constraints for the LP without the need to construct another task P', which in the case of Haslum increases exponentially in size, or in the case of Keyder, Hoffmann, and Haslum has conditional effects.

Let us illustrate the idea of dynamic merging with the logistics example introduced in Figure 1, and whose solution to the base LP is highlighted in Figure 3. The two actions in the LP solution are $\text{LOAD}(P,T,1)$ and $\text{UNLOAD}(P,T,2)$, which are indicated using bold labels and thicker arcs. The heuristic estimate for the initial state is 2, yet the minimum distance to the goal is 4 as the truck would have to drive to location 1 to load the package and drive back to location 2 to unload it. The weak estimate is caused by the prevail conditions of the actions $\text{LOAD}(P,T,1)$ and $\text{UNLOAD}(P,T,2)$, and because the goal does not impose any constraint on the final location of the truck. One way to boost the estimate is to merge the two variables into a single variable with a domain of size 6 (cf. Figure 4).

Dynamic merging allows us to explicate selected values of a merged variable without the need to explicitly form the cross product of the variables. For example, we may choose to explicate the value $\langle pos(package) = 1, pos(truck) = 1\rangle$ of the merged variable $\langle pos(package), pos(truck)\rangle$. To do so, we extend the LP with a constraint corresponding to the flow balance of the merged value (merge for short) being explicated. The constraint takes into account the producers and consumers of the merge; e.g., the action $\text{LOAD}(P,T,1)$ consumes the merge while $\text{UNLOAD}(P,T,1)$ produces it. However, we also need to keep track of actions that *could* produce or consume the merge. These *potential producers and consumers* are naturally captured in the parallel composition of the DTGs as they appear as multiple transitions with the same action label (cf. Figure 4). Under dynamic merging the product is only partially explicated by creating (using Equations 1–3) the nodes and transitions in the product DTG that are incident on the merge. For example, when the merge $\langle pos(package) = 1, pos(truck) = 1\rangle$ is explicated, only the merge and the values $\langle pos(package) = T, pos(truck) = 1\rangle$ and $\langle pos(package) = 1, pos(truck) = 2\rangle$ in the product are created, along with the transitions between them. Of these transitions, $\text{LOAD}(P,T,1)$ (resp. $\text{UNLOAD}(P,T,1)$) consumes (resp. produces) the merge as it destroys (resp. creates) the merge once applied. On the other hand, the action $\text{DRIVE}(T,2,1)$ (resp. $\text{DRIVE}(T,1,2)$) is a potential producer (resp. consumer) as it creates (resp. destroys) the merge only when it is applied at a state that does not satisfy (resp. satisfies) the merge.

In the LP formulation, the flow balance equation for the merge is defined as before by Equation 4, except that the transitions for the potential producers/consumers are associated with new variables that are called *action copies*. In the example, there is an action copy for each transition labeled with a drive action in Figure 4. These new variables however *do not vary freely* in the LP as they are linked to the original action variables in the problem. This ensures that each execution of an action copy is linked to an execution of the original action. Indeed, if $\text{Copies}(a, Z)$ is the set of copies for action a in the product DTG for Z, the *link constraint for a in Z* is:

$$f(a) \;\geq \sum_{a' \in \text{Copies}(a,Z)} f(a')\,.$$

As an illustration, let us explicate the merge $m_1 = \langle pos(package) = 1, pos(truck) = 1 \rangle$ in the example. First, a partial DTG for the product of the DTGs for the variables $pos(package)$ and $pos(truck)$ is created, along with the copies $d_{2,1}$ and $d_{1,2}$ for the actions $\textsc{Drive}(T, 2, 1)$ and $\textsc{Drive}(T, 1, 2)$ respectively. Second, the LP is extended with the flow balance equation for the merge:

$$LB_m \leq f(\textsc{Unload}(P, T, 1)) + f(d_{2,1}) - f(\textsc{Load}(P, T, 1)) - f(d_{1,2}) \leq UB_m .$$

Finally, the LP is extended with the link constraints:

$$f(\textsc{Drive}(T, 1, 2)) \geq f(d_{1,2}) ,$$
$$f(\textsc{Drive}(T, 2, 1)) \geq f(d_{2,1}) .$$

If we explicate $m_2 = \langle pos(package) = T, pos(truck) = 2 \rangle$ after m_1, the solution of the LP gives the perfect value of 4 as it includes the load and unload actions for the package plus the two drive actions that are needed in the plan. The load and unload actions appear because the net flow on $p = pos(package) = 2$ must be 1: $\textsc{Unload}(P, T, 2)$ gets inserted as it produces p while $\textsc{Load}(P, T, 1)$ appears as it produces $pos(package) = T$ which is consumed by the unload action. On the other hand, the load action consumes m_1 which is not true initially. Thus, the copy $d_{2,1}$ of $\textsc{Drive}(T, 2, 1)$ that produces m_1 gets into the solution, and similarly for the copy of $\textsc{Drive}(P, 1, 2)$ that produces m_2, as m_2 is consumed by $\textsc{Unload}(P, T, 2)$.

In summary, the basic operation when implementing dynamic merging is to merge two atoms p and q for different variables into the atom $merge(p, q)$. This operation partially creates or extends the DTG for the product of the two variables, creates the flow balance equation for the merge, and creates or updates the link constraints for the action copies that are incident on the merge. The link constraint for an action a gets updated when performing the dynamic merging of p and q when new copies for a are created.

If $\mathcal{M}$ is a set of merges, we write $h^{\mathcal{M}}_{\text{base}}$ to denote the heuristic that results of explicating all the merges in $\mathcal{M}$. It follows that dynamic merging can be done efficiently and always results in admissible estimates:

Theorem 3. *Let $\mathcal{M}$ be a set of merges. Then, the set of constraints for $\mathcal{M}$ is admissible and $h^{\mathcal{M}}_{base}$ is admissible. Further, the resulting LP has at most a linear increase (in the size of $|\mathcal{M}|$): at most nm new variables (action copies) and $n(m + 1)$ constraints are added to the LP where $n = |\mathcal{M}|$ and $m = |A|$.*

Informally, the admissibility of $h^{\mathcal{M}}_{\text{base}}$ follows from the admissibility of h_{base}. Note that, constraints in h_{base} are flow balance constraints on each atom p and constraints obtained from explicating all the merges are flow balance constraints on atoms $merge(p, q) \in \mathcal{M}$, which are both satisfied by any feasible plan. The set of merges $\mathcal{M}$ introduces a number of variables and constraints. Each merge in $\mathcal{M}$ introduces at most one action copy variable for each action, one flow balance constraint, and at most one constraint for each action linking the action copies with the original action.

Dynamic merging is a lossless operation as a full merge can be computed through a sequence of dynamic merges:

Theorem 4. *Let X and Y be two variables and $\mathcal{M}$ be the set of merges for every value of the variable XY. Then, $h^{\mathcal{M}}_{base}[P] = h_{base}[P_{XY}]$ where $h^{\mathcal{M}}_{base}[P]$ is the $h^{\mathcal{M}}_{base}$ heuristic for task P and $h_{base}[P_{XY}]$ is the h_{base} heuristic for task P_{XY}, which is P extended with the variable XY.*

In the rest of this section, we describe a simple domain-independent strategy for merging, and present some results.

A Simple Merge Strategy

The two key questions for merging are: when to merge? and what to merge? Here we adopt a simple merge strategy that (1) only performs dynamic merging at the root node of the A* search tree and (2) only considers merges between pairs of atoms, where one atom is a prevail condition and the other atom is a precondition of an action. We recognize that this strategy is by no means comprehensive and believe that there is ample opportunity to try other, more involved, merge strategies in future work.

Our merge strategy is as follows. We create the base model at the root node of the A* search tree and solve it. For each unmarked action a that appears in the solution we check if the action has any prevail conditions. If so, we mark the action and explicate $merge(p, q)$ for all atoms $p \in \text{Prev}(a)$ and $q \in \text{Pre}(a)$. In this way the prevail conditions that do not play a role in the constraints become active. After all such merges are explicated, a new solution for the LP is computed and the whole process is repeated until the solution contains no unmarked actions with prevail conditions. Since at least one action is marked at each iteration, this process terminates in at most $|A|$ iterations.

Consider the example shown in Figure 3 whose solution is $f(\textsc{Load}(P, T, 1)) = 1$ and $f(\textsc{Unload}(P, T, 2)) = 1$. Both these actions have prevail conditions which provide candidates for merging. The first action has prevail $pos(truck) = 1$ and precondition $pos(package) = 1$, while the second has prevail $pos(truck) = 2$ and precondition $pos(package) = T$. Hence, the simple merge strategy explicates the merges $m_1 = merge(1, 1)$ and $m_2 = merge(T, 2)$ (cf. Figure 4) yielding the perfect heuristic.

Putting All Together: Landmarks and Merges

Before combining the improvements given by the constraints for landmarks and the simple merge strategy, we evaluate the benefits of the latter with respect to the base model. The column $f01$ in Table 4 shows coverage results for the base LP improved with the simple strategy for dynamic merging. As can be seen, $f01$ provides a significantly higher coverage and, in some domains, the increase in the number of problems solved is significant; e.g., gripper, logistics, mprime and woodworking. In gripper, for example, $f01$ is able to solve all the instances in a backtrack free manner as the resulting heuristic is perfect. Scatter plots comparing the number of expanded nodes and the values at the initial state for the base LP and $f01$ are shown in Figure 5. In some cases $f01$ does not result in an improvement over the base LP, which is often due to the presence of actions with no prevail conditions in the initial solution making the simple merge strategy to terminate without updating the base

Domain	LM-cut	$h^*_{\text{LM-cut}}$	base	$f01$	$f10$	$f11$	$f20$	$f21$
airport (50)	**28**	**28**	20	22	26	22	25	22
barman-opt11-strips (20)	**4**	**4**	**4**	—	**4**	—	**4**	—
blocks (35)	28	28	28	28	28	28	**29**	**29**
depot (22)	**7**	**7**	**7**	6	**7**	5	**7**	5
driverlog (20)	13	13	11	**15**	11	**15**	13	**15**
elevators-opt08-strips (30)	**22**	19	9	21	9	21	18	21
elevators-opt11-strips (20)	**17**	16	7	**17**	7	**17**	15	**17**
floortile-opt11-strips (20)	**6**	**6**	4	2	4	2	**6**	5
freecell (80)	15	15	35	33	**47**	28	27	29
grid (5)	**2**	**2**	1	**2**	**2**	**2**	1	**2**
gripper (20)	7	6	6	**20**	7	**20**	5	**20**
logistics00 (28)	20	20	15	**22**	14	**22**	20	**22**
logistics98 (35)	6	6	2	7	3	7	6	**10**
miconic (150)	**141**	**141**	50	58	57	140	140	**141**
mprime (35)	22	22	18	24	20	24	20	**25**
mystery (30)	19	18	15	**20**	16	**20**	16	17
nomystery-opt11-strips (20)	**14**	**14**	10	**14**	8	**14**	12	**14**
openstacks-opt08-strips (30)	**20**	17	15	10	12	10	15	10
openstacks-opt11-strips (20)	**15**	12	7	5	7	5	7	5
openstacks-strips (30)	7	7	7	7	**8**	7	7	5
parcprinter-08-strips (30)	18	18	28	**30**	28	**30**	29	**30**
parcprinter-opt11-strips (20)	13	13	**20**	**20**	**20**	**20**	**20**	**20**
parking-opt11-strips (20)	**2**	1	1	1	1	1	1	1
pathways-noneg (30)	**5**	**5**	4	4	4	4	**5**	**5**
pegsol-08-strips (30)	27	27	**28**	**28**	26	26	27	27
pegsol-opt11-strips (20)	17	17	**18**	**18**	16	16	17	17
pipesworld-notankage (50)	**17**	16	15	13	16	15	11	13
pipesworld-tankage (50)	**11**	9	10	10	10	10	9	10
psr-small (50)	49	49	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	**7**	**7**	6	6	6	**7**	**7**	**7**
satellite (36)	**7**	**7**	6	6	6	6	**7**	**7**
scanalyzer-08-strips (30)	**15**	13	13	12	13	11	11	11
scanalyzer-opt11-strips (20)	**12**	10	10	9	9	8	8	8
sokoban-opt08-strips (30)	**28**	**28**	16	18	20	20	27	27
sokoban-opt11-strips (20)	**20**	**20**	15	15	16	16	**20**	19
tidybot-opt11-strips (20)	**13**	**13**	5	1	11	1	8	1
tpp (30)	6	6	8	**11**	8	**11**	8	10
transport-opt08-strips (30)	**11**	**11**	10	10	10	10	**11**	**11**
transport-opt11-strips (20)	**6**	**6**	**6**	5	5	5	**6**	**6**
trucks-strips (30)	10	10	9	10	9	**12**	9	11
visitall-opt11-strips (20)	10	10	17	17	17	17	**19**	**19**
woodworking-opt08-strips (30)	16	16	12	25	14	**28**	20	**28**
woodworking-opt11-strips (20)	11	11	7	18	9	**20**	15	**20**
zenotravel (20)	12	12	9	**13**	9	**13**	11	**13**
Total (1396)	756	736	594	683	630	766	749	**785**

Table 4: Coverage for LM-cut, optimal cost partitioning, and several variants of the flow-based heuristic. Variants $f1x$ and $f2x$ refer to addition of landmark constraints, and variants $fx1$ refer to the use of dynamic merging. Best results are highlighted.

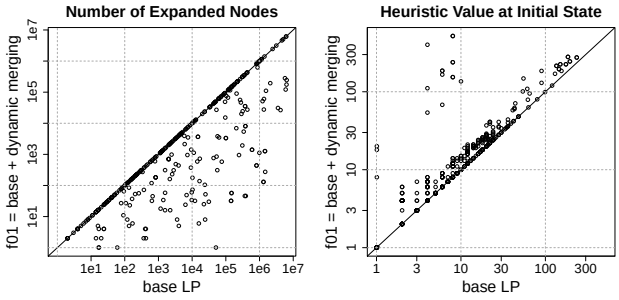

Figure 5: Number of expanded nodes and value at initial state for base LP and $f01$ across common solved tasks.

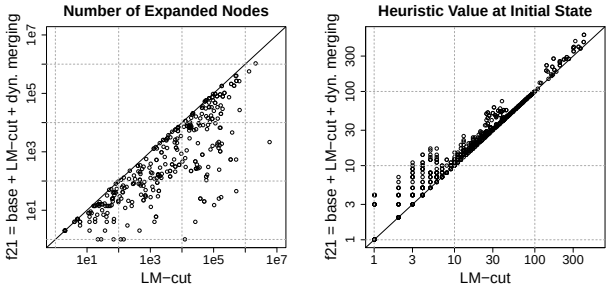

Figure 6: Number of expanded nodes and value at initial state for LM-cut and $f21$ across common solved tasks.

LP. This shortcoming, however, is due to the merge strategy and not on the general idea of dynamic merging.

The table also contains results for the optimal cost partitioning for the LM-cut landmarks in column $h_{\text{LM-cut}}$. This heuristic dominates LM-cut but, as seen, does not improve its coverage because it requires more time to compute.

Our final experiments show the improvements achieved with dynamic merging over different flow-based heuristics. These experiments correspond to the columns marked with $fx1$ that extend the heuristics for the columns marked with $fx0$ by using the simple merge strategy. The best heuristic is the one for the column $f21$ that corresponds to base LP formulation extended with the constraints for the LM-cut landmarks and the constraints obtained by using the merge strategy. As can be seen, the overall coverage for $f21$ jumps by 36 from 749 to 785 with respect to $f20$, by 102 with respect to $f01$, and by 29 with respect to LM-cut. As said above, these experiments correspond to LPs that only contain lower bounding constraints of the flows for each atom or merge as the upper bounding constraints are redundant (Pommerening et al. 2014). However, this redundancy only holds when the constraints for all the atoms in any given DTG appear in the LP, something that does not hold when performing dynamic merging. We thus conducted another experiment in which the lower and upper bounding constraints for flows are added to the LP, and obtained a coverage for the $f21$ heuristic of 798 problems solved, which is 13 more problems than the coverage of 785 reported in Table 4.

Scatter plots showing number of expanded nodes and the heuristic value at the initial state for commonly solved problems with respect to LM-cut are shown in Figure 6, and a plot comparing the solution time is shown in Figure 7.

While the number of expanded nodes for $f21$ is less than for LM-cut, it typically requires more time. This seems to indicate that spending more time to calculate a stronger heuristic is beneficial for this set of benchmarks. Note, however, that this result is dependent on the time cutoff, which is currently set at 30 minutes (the same time cutoff that has been used in the past international planning competitions).

Conclusions and Future Work

We have shown how a basic flow-based heuristic for optimal planning can be strengthened substantially by using three simple ideas: (1) the automatic reformulation of the input

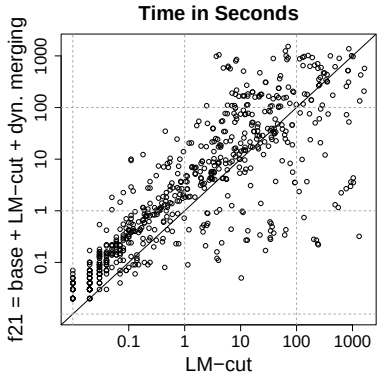

Figure 7: Time spent by A* for finding a solution when using LM-cut and $f21$ across common solved tasks.

problem to extend the goal description, (2) the addition of constraints for actions landmarks, and (3) performing detailed partial merging, that we call dynamic merging. The result is a very strong admissible heuristic that is competitive with the state of the art for optimal planning.

The first two ideas were suggested by Bonet (2013) in order to cope with goal descriptions involving few atoms and the prevail conditions that do not play an active role in the base LP formulation, yet Bonet does not provide a hint on how to do the reformulation in a domain-independent manner, and does not evaluate the impact of the information contained in the landmarks. The idea of dynamic merging is novel, but it does shows similarities with some of the recent syntactic transformations aimed at boosting standard heuristics (Haslum 2012; Keyder, Hoffmann, and Haslum 2012). Implementing a domain-independent and successful strategy for dynamic merging is quite a challenge. We tried a simple strategy that only considers the prevail conditions of the actions that become active in the solution of the LP, but it is clear that there are other opportunities for developing more complex and better strategies. Likewise, in this work, we only perform dynamic merging on atoms p and q that belong to the original problem and do not attempt to build more complex merges involving three or more atoms. We believe that flow-based heuristics provide a rich area for future research and important advances in the field. In particular, we plan to extend this work by designing better and more powerful merge strategies that may include the formation of complex merges. On the theoretical side, it is an open question to identify the real power of flow-based heuristics in relation to other heuristics, yet a first result in this direction is given by Pommerening et al. (2014).

Acknowledgements

Experiments were performed in the DTIC's HPC cluster of the Universitat Pompeu Fabra, Barcelona, Spain.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. ECAI*, 329–334. Lisbon, Portugal: IOS Press.

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In *Proc. IJCAI*, 2268–2274.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In *Proc. SPIN Workshop*, 19–34. Vienna, Austria: Springer: LNCS 3925.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP*, 13–24. Toledo, Spain: Springer: LNCS.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. AAAI*, 1163–1168. Pittsburgh, PA: AAAI Press / MIT Press.

Haslum, P. 2012. Incremental lower bounds for additive cost planning problems. In *Proc. ICAPS*, 74–82. Sao Paulo, Brazil: AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS*, 162–169. Thessaloniki, Greece: AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS*, 176–183. Providence, RI: AAAI Press.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI*, 1728–1733. Pasadena, CA: AAAI Press.

Katz, M., and Domshlak, C. 2010. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12–13):767–798.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In *Proc. ICAPS*, 128–136. Sao Paulo, Brazil: AAAI Press.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for And/Or graphs. In *Proc. ECAI*, 335–340.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. Lp-based heuristics for cost-optimal planning. In *Proc. ICAPS*. Portsmouth, NH: AAAI Press. To appear.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI*, 2357–2364. Beijing, China: AAAI Press.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. CP*, 651–665. Springer: LNCS 4741.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS Doctoral Consortium*, 156–160.