Additive Merge-and-Shrink Heuristics for Diverse Action Costs

Gaojian Fan, Martin Müller and Robert Holte
University of Alberta, Canada
{gaojian, mmueller, rholte}@ualberta.ca

Abstract

In many planning applications, actions can have highly diverse costs. Recent studies focus on the effects of diverse action costs on search algorithms, but not on their effects on domain-independent heuristics. In this paper, we demonstrate there are negative impacts of action cost diversity on merge-and-shrink (M&S), a successful abstraction method for producing high-quality heuristics for planning problems. We propose a new cost partitioning method to address the negative effects of diverse action costs on M&S. We investigate non-unit cost IPC domains, especially those for which diverse action costs have severe negative effects on the quality of the M&S heuristic. Our experiments demonstrate that in these domains, an additive set of M&S heuristics using the new cost partitioning method produces much more informative and effective heuristics than creating a single M&S heuristic which directly encodes diverse costs.

1 Introduction

Diverse action costs are common in many real-world planning problems. Typical examples are logistics-like domains, in which loading or unloading a package is cheaper than moving a vehicle, or where the cost of moving a vehicle varies widely with the distance between locations. Another set of examples are manufacturing applications, where different types of processing incur a diverse set of costs.

There have been several studies of the effects of action cost diversity on heuristic search [Cushing *et al.*, 2010; Aghighi and Bäckström, 2015; 2016; Fan *et al.*, 2017], and improved search algorithms designed to address diverse action costs [Thayer and Ruml, 2011; Wilt and Ruml, 2011; 2014]. However, to the best of our knowledge, there is no study on the influence of diverse action costs on the construction and performance of domain-independent heuristics in the planning literature.

Merge-and-shrink (M&S) is an abstraction method that can produce high-quality abstraction heuristics [Helmert *et al.*, 2014]. While M&S heuristics can be computed for general non-unit action costs, little is known about how action cost diversity affects this abstraction method. In this paper, we study the impact of action cost diversity on the merge-and-shrink process, and develop an improved algorithm, called DCP-MS, which computes additive M&S heuristics for diverse action costs.

After a brief discussion of background in Section 2, we show in Section 3 that the M&S heuristic suffers from diverse action costs on several domains from the International Planning Competition[1] (IPC), compared to their unit-cost counterparts. The section includes in-depth analysis of experimental results. Motivated by these results, in Section 4 we propose a new cost partitioning method, *delta cost partitioning*, which limits the cost diversity of partitioned cost functions to only two costs so that M&S can process them more effectively.

The cost partitioning exploits the power of M&S for unit costs to help improve the quality of the heuristic for the non-unit cost case. Our experiments in Section 5 show that an additive set of M&S abstractions using delta cost partitioning can produce much more informed heuristics than a single M&S heuristic that encodes the diverse action costs directly, and the benefits are most pronounced in domains where action cost diversity has extremely negative effects on M&S.

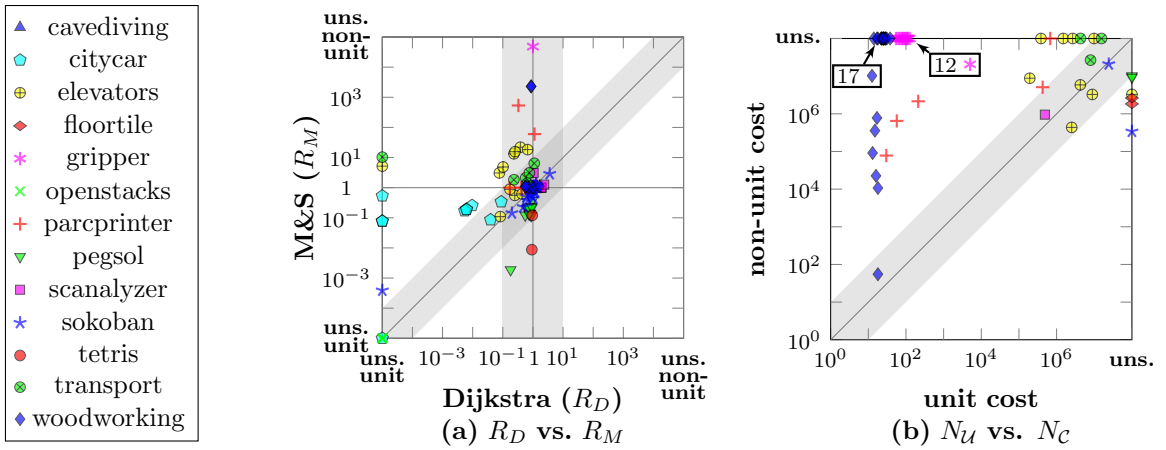
2 Background

A *transition graph* is a quintuple $\mathcal{S} = \langle S, L, T, s_{\text{init}}, S_* \rangle$ where the state space S is a finite set of states, L is a finite set of transition labels, and $T \subseteq S \times L \times S$ is a set of labelled transitions. $s_{\text{init}} \in S$ is an initial state and $S_* \subseteq S$ is a set of goal states. An *abstraction* α is a mapping from S to an abstract state space, which induces an *abstract transition graph* $\alpha(\mathcal{S}) = \langle \alpha(S), L, \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}, \alpha(s_{\text{init}}), \alpha(S_*) \rangle$.

Let $\mathcal{S} = \langle S, L, T, s_{\text{init}}, S_* \rangle$. A cost function is a mapping $\mathcal{C} : L \mapsto \mathbb{R}_0^+$, where $\mathbb{R}_0^+$ is the set of non-negative real numbers. For a cost function $\mathcal{C}$, the distance from state s to state t in $\mathcal{S}$ is the cost of a least-cost path from s to t, where a path is a sequence $(s_0, l_1, s_1, ..., s_{n-1}, l_n, s_n)$ such that $n \in \mathbb{N}_0$ and $(s_{i-1}, l_i, s_i) \in T$ for $i \in \{1, 2, ..., n\}$, and its cost is $\sum_{j=1}^{n} \mathcal{C}(l_j)$. The goal distance $h^*(s)$ of state s is the distance from s to a nearest goal state.

A heuristic function $h : S \mapsto \mathbb{R}_0^+$ maps each state to a non-negative value or infinity. Heuristic h is admissible if $h(s) \leq$

[1] www.icaps-conference.org/index.php/Main/Competitions



(a) R_D vs. R_M (b) $N_{\mathcal{U}}$ vs. $N_{\mathcal{C}}$

Figure 1: (a) R_D vs. R_M on instances solved by both A* with M&S and Dijkstra's algorithm; (b) $N_{\mathcal{U}}$ vs. $N_{\mathcal{C}}$ on instances that can only be solved by A* with M&S.

$h^*(s)$. For a cost function $\mathcal{C}$, an abstraction α provides an abstraction heuristic $h^{\mathcal{C}}_{\alpha}$ in which the heuristic value of s in S is the goal distance of $\alpha(s)$ in the abstract transition graph $\alpha(\mathcal{S})$. Any abstraction heuristic is admissible.

Merge-and-shrink (M&S) [Helmert *et al.*, 2014] builds an abstraction by transforming a set of abstract transition graphs into a single one. The transformations consist of *merging* two abstract transition graphs into one, *shrinking* the size of an abstract transition graph by combining multiple states into one state, and *label reduction* which maps different transition labels into one label. Label reduction is essential for M&S to compactly represent *bisimulation abstractions* which capture the goal distance $h^*(s)$ for all states s [Nissim *et al.*, 2011].

For a cost function $\mathcal{C}$ on L, a cost partitioning of $\mathcal{C}$ is a set of cost functions $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_n$ on L, such that $\sum_{i=1}^{n} \mathcal{C}_i(l) \leq \mathcal{C}(l)$ for any $l \in L$. A set of heuristics h_1, h_2, ..., h_n is additive if $h_{\text{sum}}(s) = \sum_{i=1}^{n} h_i(s)$ is an admissible heuristic. For any transition graph $\mathcal{S}$, cost function $\mathcal{C}$, cost partitioning $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_n$ of $\mathcal{C}$, and set of abstractions α_1, α_2, ..., α_n for $\mathcal{S}$, the set of abstraction heuristics $h^{\mathcal{C}_1}_{\alpha_1}$, $h^{\mathcal{C}_2}_{\alpha_2}$, ..., $h^{\mathcal{C}_n}_{\alpha_n}$ is additive [Yang *et al.*, 2008].

3 Action Cost Diversity and M&S Heuristics

There have been several studies on the effects of diverse action costs on search, but none of them focuses on the effects on domain-independent heuristics like M&S. In this section, we first verify experimentally that diverse action costs have negative effects on the merge-and-shrink heuristic compared to the unit cost, then discuss how diverse action costs could influence M&S abstraction construction.

3.1 Experimental Verification

To determine whether action cost diversity is indeed a problem for planning with M&S, we ran experiments on both unit cost and non-unit cost versions of a set of IPC domains with and without M&S heuristics. Our test set contains the 12 non-unit cost IPC domains, as well as the unit cost domain GRIPPER. We include GRIPPER because M&S can build bisimulations for all instances in this domain [Nissim *et al.*, 2011]. We build the unit cost version for each non-unit cost IPC instance, as well as a non-unit action cost version of GRIPPER instances as follows: the `move` action still costs 1, but `pick` and `drop` actions for ball b now have cost $(b \bmod n_c) + 1$, where $n_c = 4$ in this experiment.

Relative Effects of Cost Diversity on M&S Heuristics

For any planning instance, let $\mathcal{U}$ and $\mathcal{C}$ be the unit cost function and the non-unit cost function respectively. We ran A* with the default M&S heuristics[2] separately for $\mathcal{U}$ and $\mathcal{C}$ for each of our instances. Each run has a 30 minute time limit and a 2 GB memory limit. For each instance we compute the ratio $R_M = N_{\mathcal{C}}/N_{\mathcal{U}}$, where $N_{\mathcal{C}}$ and $N_{\mathcal{U}}$ are the numbers of node expansions by A* with M&S heuristics using $\mathcal{C}$ and $\mathcal{U}$ respectively. R_M is the y-axis of Figure 1(a) (the x-axis is defined below, for now we are only considering R_M, the height of each point in Figure 1(a)). The plot uses a log scale and $R_M = 1.0$ is the horizontal line in the middle of the plot. Points above this line represent instances for which $R_M > 1.0$, i.e. A* with M&S expands more nodes (performs worse) with $\mathcal{C}$ than it does with $\mathcal{U}$. Points below the line are instances on which A* with M&S performs better with $\mathcal{C}$ than with $\mathcal{U}$. If an instance is solved using $\mathcal{C}$ but not $\mathcal{U}$ it is placed at the bottom of the plot (on the line labelled **uns. unit**). There are no instances solved using $\mathcal{U}$ but not $\mathcal{C}$ in this plot. Instances that are unsolved with both $\mathcal{U}$ and $\mathcal{C}$ are not shown in the plot.

Changing the action costs can affect search performance even without heuristics. The instances with $R_M > 1.0$ may be simply harder to solve with $\mathcal{C}$ than with $\mathcal{U}$ even without M&S. In order to separate the effects of diverse costs on the M&S heuristics from their effects on search without heuristics we solved all the instances again, with both cost functions, without a heuristic (Dijkstra's algorithm, $f = g$), with the same 30min/2GB limit per instance. Analogous to R_M we define R_D to be the ratio of the number of nodes Dijk-

[2]The recommended configuration: DFP merging, bisimulation shrinking, label reduction before shrinking, maximum of 50,000 states per abstraction (from `www.fast-downward.org/Doc/Heuristic#Merge-and-shrink_heuristic`).

Region	R_D	$\frac{R_M}{R_D}$
< 1	61	41
< 0.1	17	3
> 1	36	53
> 10	0	19

Table 1: Numbers of instances in specific regions of Figure 1(a).

$\mathcal{C}$	c_1	c_2	c_3	$\cdots$	c_{n-1}	c_n
$\mathcal{C}_1$	Δ_1	Δ_1	Δ_1	$\cdots$	Δ_1	Δ_1
$\mathcal{C}_2$	0	Δ_2	Δ_2	$\cdots$	Δ_2	Δ_2
$\mathcal{C}_3$	0	0	Δ_3	$\cdots$	Δ_3	Δ_3
$\vdots$				$\vdots$		
$\mathcal{C}_n$	0	0	0	$\cdots$	0	Δ_n

Table 2: The cost mapping of delta cost partitioning.

$\mathcal{C}$	1	3	10
$\mathcal{C}_1$	1	1	1
$\mathcal{C}_2$	0	2	2
$\mathcal{C}_3$	0	0	7

Table 3: DCP of a cost function $\mathcal{C}$ that have three different costs 1, 3 and 10.

stra's algorithm expands using $\mathcal{C}$ to the number of nodes it expands using $\mathcal{U}$. This is the x-axis in Figure 1(a). Like the y-axis it is a log-scale, $R_D = 1.0$ is the vertical line in the middle of the plot, and instances solved using $\mathcal{C}$ but not $\mathcal{U}$ are placed on the left edge of the plot (on the line labelled **uns. unit**). The vertical gray zone indicates $10^{-1} \leq R_D \leq 10$. Note that Dijkstra's algorithm performs better with $\mathcal{C}$ than with $\mathcal{U}$ (more instances with $R_D < 1$ than $R_D > 1$ and more with $R_D < 10^{-1}$ than $R_D > 10$), which has been observed before [Fan *et al.*, 2017].

The points above the diagonal line $y = x$ are instances for which $R_M > R_D$, meaning that compared to Dijkstra, the M&S heuristic performs worse with $\mathcal{C}$ than with $\mathcal{U}$. For such instances, the M&S heuristic may reduce the number of node expansions for both $\mathcal{C}$ and $\mathcal{U}$, but reduce it proportionally less for $\mathcal{C}$ than for $\mathcal{U}$. For instances below the line $y = x$, M&S performs worse using $\mathcal{U}$ than $\mathcal{C}$. For a point close to the diagonal line, the M&S heuristic shows no obvious advantage for either $\mathcal{C}$ or $\mathcal{U}$. The diagonal gray zone indicates the difference within the factor of 10.

Table 1 gives the number of instances in specific regions of Figure 1(a). Dijkstra's algorithm expands more nodes using $\mathcal{U}$ than $\mathcal{C}$ on more instances (61 instances for $R_D < 1$ vs. 36 for $R_D > 1$ in column "R_D"). In contrast, the M&S heuristic shows inferior performance when using $\mathcal{C}$ instead of $\mathcal{U}$ compared to Dijkstra's algorithm (41 instances for $R_M < R_D$ but 53 for $R_M > R_D$, in column "R_M/R_D"). 19 instances have a more than 10-fold increase in the ratio of node expansions from using $\mathcal{C}$ compared to using $\mathcal{U}$, while only 3 instances have a more than 10-fold decrease.

Newly Solved Instances with M&S

A handful of instances that cannot be solved with either $\mathcal{U}$ or $\mathcal{C}$ by Dijkstra's algorithm can be solved by A* with M&S heuristics. These instances are omitted from Figure 1(a) as they do not have a meaningful value of R_D. We show these instances in Figure 1(b), which compares their $N_{\mathcal{U}}$ (x-axis) and $N_{\mathcal{C}}$ (y-axis).

Figure 1(b) shows many more instances above the diagonal line than below. The instances above the gray zone ($N_{\mathcal{C}} > 10N_{\mathcal{U}}$) or on the top **uns.** line are those solved by A* with M&S using $\mathcal{U}$ but require orders of magnitude more node expansions or cannot be solved within memory/time limits using $\mathcal{C}$. There are 46 such instances in total, from 5 domains: WOODWORKING, GRIPPER, ELEVATORS, PARCPRINTER and TRANSPORT. On the other hand, there are only 6 instances for which A* with M&S performs much better using $\mathcal{C}$ than using $\mathcal{U}$ (on the **uns.** line on the right edge of Figure 1(b)).

Results in Figure 1 and Table 1 show that negative effects outweigh positive ones for action cost diversity on A* with the M&S heuristic. Diverse action costs have an extreme negative effect on WOODWORKING and GRIPPER, while ELEVATORS, PARCPRINTER and TRANSPORT are affected negatively to a more moderate degree. Neither $\mathcal{U}$ nor $\mathcal{C}$ have a clear advantage in domains CAVEDIVING, SCANALYZER, TETRIS[3] and OPENSTACKS. $\mathcal{C}$ beats $\mathcal{U}$ in domains PEGSOL, FLOORTILE and SOKOBAN by a small margin. CITYCAR is a special domain because its instances are easier to solve with $\mathcal{C}$ than with $\mathcal{U}$, both with and without M&S, but the M&S heuristics reduce the search effort more for $\mathcal{U}$ than for $\mathcal{C}$, relative to Dijkstra's algorithm.

3.2 Building a M&S Heuristic with Diverse Costs

In M&S, label reduction [Sievers *et al.*, 2014] facilitates bisimulation shrinking [Nissim *et al.*, 2011] in reducing abstraction sizes without information loss in heuristics. It is a critical, often necessary (e.g., for GRIPPER), technique for M&S to create compact bisimulations. Action cost diversity has a direct impact on label reduction. Current label reduction techniques are *cost-exact*: to avoid information loss, only labels with the same cost can be reduced to a single label. With unit costs, cost-exactness is guaranteed trivially. With non-unit costs, label reductions are much more limited. With more distinct labels in a transition system, bisimulation abstractions become larger, and more harmful shrinking operations are required to limit the abstraction size.

Action cost diversity affects M&S not only through label reduction, but also through changes in merging and shrinking decisions. Non-unit and unit cost induce different distributions of states regarding their distances from the abstract initial state and to the abstract goal. Merging and shrinking decisions are often based on such distance information. For example, the state-of-the-art merging and shrinking strategies [Dräger *et al.*, 2006; Nissim *et al.*, 2011] tend to prioritize regions close to the abstract goal, and thus heuristics are more informed in these regions than elsewhere. In this paper, we do not address such issues, and leave them as promising directions for future studies.

4 Cost Partitioning for Diverse Action Costs

In this section, we propose a cost partitioning method, called *delta cost partitioning (DCP)*, for M&S to handle diverse ac-

[3] 5 TETRIS instances were not solved by M&S method with either $\mathcal{C}$ or $\mathcal{U}$, but solved by Dijkstra's algorithm because M&S cannot build the abstraction within the time limit for these instances. Since this is not a cost-related problem, we omit these instances.

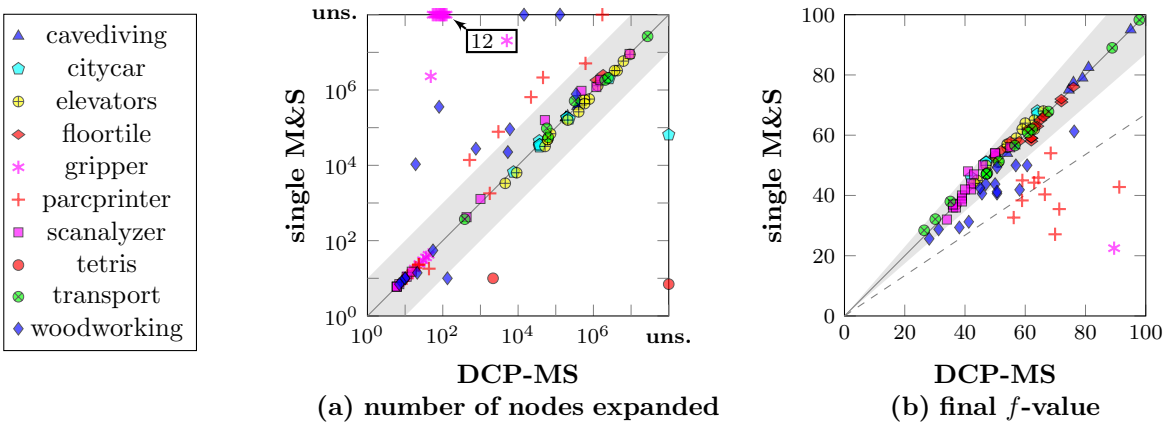


Figure 2: Compare DCP-MS (x-axes) and single M&S (y-axes) on: (a) Numbers of node expansions; (b) The final f-value before time/memory limit is reached of unsolved instances (but with M&S abstractions built successfully).

tion costs better. As M&S benefits from the simplicity of unit cost in many cases, we partition action costs into cost functions that are as simple as possible.

In the remainder of this section, let L be the label set of a planning task, and $\mathcal{C}$ be any cost function on L. Let $c_0 = 0$ and $0 < c_1 < c_2 < \cdots < c_n$ be the n different positive cost values to which the labels in L are mapped by $\mathcal{C}$, and $\Delta_i = c_i - c_{i-1}$ for $i \in \{1, 2, ..., n\}$. Let $L_i = \{l \mid \mathcal{C}(l) = c_i\}$ be the set of labels that have cost c_i for $i \in \{0, 1, ..., n\}$. DCP divides the costs $c_1, c_2, \cdots, c_n$ among n *delta cost functions* $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n$ as follows. For $i \in \{1, 2, ..., n\}$,

$$\mathcal{C}_i(l) = \begin{cases} 0, & l \in \bigcup_{j=0}^{i-1} L_j \,, \\ \Delta_i, & l \in \bigcup_{j=i}^{n} L_j \,. \end{cases}$$

DCP maps a label to a new cost in a delta cost function depending on the label's original cost and the delta cost function. This mapping is illustrated in Table 2, and an example is shown in Table 3.

Since each delta cost function $\mathcal{C}_i$ maps a label to either cost 0 or Δ_i, it has at most two different costs. With the limited number of distinct costs in a delta cost function, label reduction becomes much less restricted. This could be extremely beneficial if label reduction is essential for constructing a high quality M&S heuristic for the planning task.

After the cost partitioning, we then run M&S on each delta cost function and obtain an additive set of M&S heuristics. In the rest of the paper, we call this method DCP-MS and use the term "single M&S" to refer to the classical use of a single M&S heuristic.

5 Experiments

In this section, we compare DCP-MS with single M&S from the following perspectives:

First, there could be complex interactions between action costs and M&S's behaviour. While cost functions with limited cost diversity seem beneficial for label reduction, we do not know how shrinking and merging strategies are affected, so the effects of delta cost partitioning on M&S need to be tested experimentally.

Second, DCP-MS incurs obvious computational overhead from constructing multiple M&S, and from multiple lookups required for the additive heuristic. We need to evaluate whether the benefit of a better heuristic outweighs the overhead.

Third, because GRIPPER is a domain where label reduction is essential but suffers from cost diversity, we perform a case study on this domain with cost functions of different cost diversity to see how DCP-MS ameliorates the problem.

Our experimental setting is the same as in Section 3.1. The M&S construction of DCP-MS uses the same configuration as single M&S (the default recommended configuration). Domains PEGSOL, SOKOBAN and OPENSTACKS are excluded here, since these domains use only action costs 0 and 1 and thus delta cost partitioning only reproduces the single original cost function. In addition, from the results in Section 3.1, these three domains are not affected negatively by the action cost diversity in their non-unit cost function, so they are not the target domains for our method.

5.1 Performance of Delta Cost Partitioning

In Figure 2(a), the numbers of nodes expanded by single M&S (y-axis) are plotted against DCP-MS (x-axis). In the gray zone, the ratio is within a factor of 10. Points above the gray zone are strongly favorable for DCP-MS, while it is much inferior for points below.

Overall, DCP-MS outperforms single M&S on many more instances, with 24 instances above the gray zone, of which 15 are solved only by DCP-MS. These instances are from the three domains GRIPPER, WOODWORKING and PARCPRINTER, which are the domains that are most affected by action cost diversity, according to the results in Section 3.1. In particular, for the 12 GRIPPER instances solved only by DCP-MS, they are solved with the minimal search effort, i.e., only states along the optimal solution are expanded. The performance of DCP-MS on the non-unit cost version of GRIPPER matches that of single M&S on the unit cost version of GRIPPER. Single M&S outperforms DCP-MS on only 4 instances. Among them, two are unsolved by DCP-MS because it times out when building the abstractions.

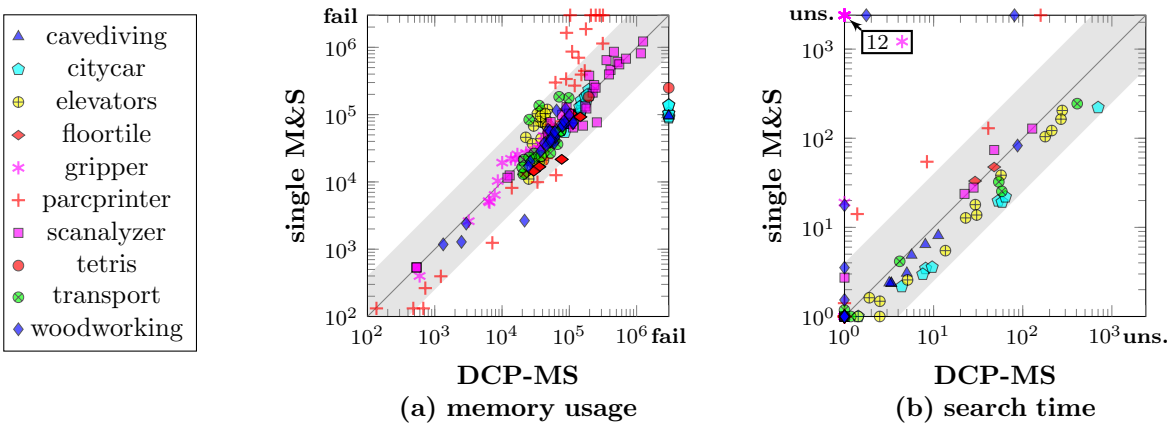


Figure 3: Compare DCP-MS (x-axes) and single M&S (y-axes) on: (a) Memory used for M&S constructions; (b) Search time (in seconds) of instances with successful M&S abstraction constructions (search time less than 1 second is treated as 1 second).

Final f-value on Unsolved Instances

A* expands an open node with the smallest f-value. This f-value, which increases as A* search progresses for consistent heuristics, indicates how much progress A* has made towards finding an optimal path to goal. For instances that are unsolved by A* with either DCP-MS or single M&S (but with M&S abstractions built successfully and A* search started), we compare their final f-values when the time/memory limit is reached in Figure 2(b).

Let f_{single} and f_{DCP} denote the largest f-value of states A* has expanded when the time/memory limit is reached, using single M&S and DCP-MS respectively. We normalize the f-values due to the wide value range across domains. A small difference between f_{single} and f_{DCP} could mean an exponential difference in numbers of node expansions, so Figure 2(b) compares f_{DCP} (x-axis) and f_{single} (y-axis) in a linear scale plot. The gray zone is defined between the lines $f_{\text{single}} = \frac{11}{10} f_{\text{DCP}}$ and $f_{\text{single}} = \frac{10}{11} f_{\text{DCP}}$. The dashed line is defined by $f_{\text{DCP}} = \frac{3}{2} f_{\text{single}}$. In Figure 2(b), there is only one instance above to the gray zone (f_{single} is at least 10% larger than f_{DCP}). In contrast, there are 21 instances below the gray zone (f_{DCP} is at least 10% larger than f_{single}), and 7 of them are even below the dashed line ($f_{\text{DCP}} > 150\% \times f_{\text{single}}$).

5.2 Computational Overhead

For the IPC domains in our experiments, a delta cost partitioning could produce between 2 and 27 delta cost functions depending on the planning instances. DCP-MS may have heavy computational overheads due to multiple M&S constructions and multiple heuristic lookups.

M&S Constructions in DCP-MS

Building a M&S abstraction can be an expensive process. However, our results show that both DCP-MS and single M&S finish the M&S constructions on most of the instances. Among the 237 instances tested in our experiments, there are only 9 instances where DCP-MS fails during the M&S construction phase due to time out but single M&S succeeds, and 5 instances where single M&S fails to build the M&S abstraction due to running out of memory but DCP-MS builds multiple M&S abstractions successfully.

Figure 3(a) compares DCP-MS (x-axis) with single M&S (y-axis) in terms of the maximal memory allocated during M&S construction. The memory usage of M&S construction for DCP-MS does not increase linearly with the number of delta cost functions. Most points are located within the gray zone (within the difference of the factor of 4), and clustered approximately evenly on both sides of the diagonal line, meaning the neither single M&S nor DCP-MS has a clear advantage in peak memory usage. On 5 PARCPRINTER instances, single M&S runs out of memory during the M&S constructions, while DCP-MS successfully builds multiple abstractions.

The key to the lower memory consumption of DCP-MS in such cases, and its good memory performance overall, is that M&S requires much less memory for the heuristic lookup tables than for abstract transition graphs. Because only the heuristic lookup tables are needed in search and DCP-MS constructs one M&S abstraction at a time for each cost function, the memory for the M&S abstract transition graphs can be released and reused after each construction.

DCP-MS fails to build abstractions while single M&S succeeds for 9 instances (right **fail** line) due to time out. Table 4 compares the M&S construction time used by DCP-MS and single M&S for instances where both methods build M&S abstractions successfully. In Table 4, column "N" shows the average number of M&S abstractions that DCP-MS has to build for each domain. The M&S construction time of DCP-MS is expected to be N times larger than that of single M&S. Column "ratio" gives the ratios of the average construction time of DCP-MS (shown in column "DCP-MS") to that of single M&S (shown in column "single"). We see that the ratio of actual time of M&S constructions for DCP-MS to that of single M&S is often less than N. The ratio is larger than N in only two domains. The 9 instances where DCP-MS times out during the M&S construction are from the domains CAVEDIVING, CITYCAR and TETRIS where building a M&S abstraction takes a relatively long time. Note that in domain PARCPRINTER, the ratio of construction time for DCP-MS to single M&S is much less than N, which means building a M&S abstraction with the original cost function takes much more time than building one with a delta cost function of re-

Domain	N	ratio	DCP-MS	single
cavediving	4.44	4.08	597.66	146.31
citycar	3.00	3.40	502.47	147.71
elevators	8.38	3.21	15.15	4.72
floortile	4.00	4.50	70.27	15.61
gripper	4.00	2.46	15.97	6.50
parcprinter	11.92	1.23	125.80	102.35
scanalyzer	2.00	1.59	147.53	92.62
tetris	3.00	1.60	1,281.99	802.71
transport	19.55	13.48	44.65	3.31
woodworking	5.80	4.04	202.39	50.12

Table 4: Comparing the M&S construction time (in second) for DCP-MS and single M&S.

duced cost diversity.

Search Time

To evaluate the overhead of multiple heuristic lookups, in Figure 3(b) we compare the search time of DCP-MS and single M&S on all instances where M&S construction succeeds. DCP-MS reduces the search time for several instances from GRIPPER, WOODWORKING and PARCPRINTER, because the reduction of numbers of node expansions by DCP-MS outweighs the overhead of multiple heuristic lookups for the three domains whose average numbers of heuristic lookups per state are 4.00, 3.30 and 12.13 respectively (instances that fail during M&S constructions are excluded).

There are instances where search takes less time with a single M&S heuristic than with the additive heuristics of DCP-MS. However, for all these instances, the search time is within the factor of 4 (in the gray zone in Figure 3(b)).

TRANSPORT and ELEVATORS instances have the two highest average numbers of heuristic lookups per state evaluation: 13.86 for TRANSPORT and 8.25 for ELEVATORS. The search time of DCP-MS on instances from these two domains certainly does not grow linearly in the number of heuristic lookups per state, as the numbers of node expansions of DCP-MS on these instances are very close to that of single M&S (see Figure 2(a)).

There are 105 instances where both methods finish building M&S abstractions but fail during search. On only 8 of these instances DCP-MS fails due to time out during search while single M&S fails due to the memory limit, and on the other 97 instances both methods fail due to the memory limit.

Overall, DCP-MS has only a small computational overhead in its M&S construction and search.

5.3 Case Study of GRIPPER

The non-unit cost function in the modified GRIPPER domain from Section 3.1 has 4 different costs. Our earlier experiments showed that single M&S has trouble dealing with such diversity, while DCP-MS works very well. In this section, we further test how well DCP-MS performs as the cost diversity increases.

The `pick/drop` action costs for ball b are $(b \bmod n_c)+1$. The parameter n_c gives an upper bound on the number of different costs for `pick/drop` actions. The number of balls in IPC GRIPPER instances ranges from 4 to 42. We compare the number of instances solved by single M&S and DCP-MS as n_c increases. In addition to the default M&S configuration, we also test the variant that does not limit abstraction size, which has the best performance on the unit cost GRIPPER.

n_c	50K		no limit	
	DCP-MS	single	DCP-MS	single
1	19	19 (19)	20	20
2	19	12 (12)	20	20
4	19	7 (6)	20	20
8	16	7 (4)	20	11
16	14	7 (4)	20	7
#balls	13	7 (4)	18	7

Table 5: Coverage of GRIPPER with increasing cost diversity for DCP-MS and single M&S using two different M&S configurations.

Table 5 shows the coverage of DCP-MS and single M&S for different n_c. For the last row of $n_c =$ "#balls", each ball has a different `pick/drop` cost. The columns under "50K" and "no limit" show the coverage of DCP-MS and single M&S using M&S configurations with and without the 50K abstraction size limit respectively. For single M&S with 50K size limit, the number of instances solved with the minimal search effort are shown in brackets. For all other methods, all solved instances are actually solved with the minimal search effort. With either M&S configuration, the coverage of single M&S decreases faster than DCP-MS as n_c increases. When every ball has a different cost, DCP-MS with no size limit can solve 18 instances with the minimal search effort, while single M&S can only solve the 7 smallest instances.

6 Related Work

To our knowledge, there is no previously published work on practical cost partitioning for additive M&S. A theoretical result by Helmert and Domshlak [2009] demonstrates that additive M&S heuristics strictly dominate the additive $h^{\max}$ [Bonet and Geffner, 2001] and landmark heuristics, in the sense that for a given state, additive M&S can provide a better heuristic. While optimal cost partitioning for abstraction heuristics can be computed in polynomial time [Katz and Domshlak, 2008], more efficient methods are needed in practice. Saturated cost partitioning [Seipp and Helmert, 2014; Seipp *et al.*, 2017] is a practical method for approximating optimal cost partitioning for abstractions.

7 Conclusion

We study the effects of diverse action costs on M&S. We show that action cost diversity can affect M&S negatively and propose a new method, DCP-MS, additive M&S with delta cost partitioning, to address this issue. Our experiments show that DCP-MS produces much more informative heuristics than the standard M&S on several IPC domains, especially on those affected negatively by action cost diversity.

References

[Aghighi and Bäckström, 2015] Meysam Aghighi and Christer Bäckström. Cost-optimal and net-benefit planning - A parameterised complexity view. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 1487–1493, 2015.

[Aghighi and Bäckström, 2016] Meysam Aghighi and Christer Bäckström. A multi-parameter complexity analysis of cost-optimal and net-benefit planning. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, pages 2–10, 2016.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Cushing *et al.*, 2010] William Cushing, J. Benton, and Subbarao Kambhampati. Cost based search considered harmful. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search*, pages 140–141, 2010.

[Dräger *et al.*, 2006] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. In *Proceedings of Model Checking Software, 13th International SPIN Workshop*, volume 3925, pages 19–34, 2006.

[Fan *et al.*, 2017] Gaojian Fan, Martin Müller, and Robert Holte. The two-edged nature of diverse action costs. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, 2017. To appear.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 162–169, 2009.

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, 61(3):16:1–16:63, 2014.

[Katz and Domshlak, 2008] Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 174–181, 2008.

[Nissim *et al.*, 2011] Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1983–1990, 2011.

[Seipp and Helmert, 2014] Jendrik Seipp and Malte Helmert. Diverse and additive cartesian abstraction heuristics. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, pages 289–297, 2014.

[Seipp *et al.*, 2017] Jendrik Seipp, Thomas Keller, and Malte Helmert. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3651–3657, 2017.

[Sievers *et al.*, 2014] Silvan Sievers, Martin Wehrle, and Malte Helmert. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2358–2366, 2014.

[Thayer and Ruml, 2011] Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 674–679, 2011.

[Wilt and Ruml, 2011] Christopher Makoto Wilt and Wheeler Ruml. Cost-based heuristic search is sensitive to the ratio of operator costs. In *Proceedings of the 4th Annual Symposium on Combinatorial Search*, pages 192–197, 2011.

[Wilt and Ruml, 2014] Christopher Makoto Wilt and Wheeler Ruml. Speedy versus greedy search. In *Proceedings of the 7th Annual Symposium on Combinatorial Search*, pages 184–192, 2014.

[Yang *et al.*, 2008] Fan Yang, Joseph C. Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research*, 32:631–662, 2008.