Fluent Merging: A General Technique to Improve Reachability Heuristics and Factored Planning

Menkes van den Briel
Department of Industrial Engineering
Arizona State University
Tempe AZ, 85287-8809
menkes@asu.edu

Subbarao Kambhampati
Department of Computer Science
Arizona State University
Tempe AZ, 85287-8809
rao@asu.edu

Thomas Vossen
Leeds School of Business
University of Colorado at Boulder
Boulder CO, 80309-0419
vossen@colorado.edu

Abstract

Fluent merging is the process of combining two or more fluents (state variables) into a single "super" fluent. By compiling in some of the inter-fluent interactions, fluent merging can (1) help improve informedness of relaxed reachability heuristics, and (2) improve the efficiency of factored planning as it removes some of the inter-variable dependencies. Although some special cases of fluent merging have been around (under other names), the technique in its full generality has not been exploited or analyzed. In this paper, we discuss the general motivations for and tradeoffs in fluent merging. We will argue that existing techniques are too conservative in identifying mergeable fluents. We will then provide some novel techniques based on causal graph analysis for identifying mergeable fluents.

Introduction

In this paper we describe fluent merging, the process of combining two or more state variables into a single "super" state variable. Fluent merging, when done judiciously, can lead to better heuristic estimates and more effective factored planning.

Let us start with an informal example to illustrate the idea of fluent merging (we shall formalize this later). In a simple Logistics problem instance with one truck, one package, and two locations, we may combine the fluents $at(truck1, loc1)$ and $at(truck1, loc2)$, which take values from $\{T, F\}$, into a super-fluent. This new fluent takes values from $\{T, F\} \times \{T, F\}$.

On the face of it, fluent merging seems like a rather quixotic idea as it runs counter to the conventional wisdom that it is advantageous to represent and reason with domains in terms of individual fluents (the so called "factored representations"). After all, the strategy of merging fluents can, in the extreme, lead us to a non-factored representation where a single super-fluent has an exponential domain size, with each domain value corresponding to each of the states in the domain.

The reason merging winds up being useful in some cases is that if we merge fluents that have strong dependencies, then their effective domain can be much smaller than the Cartesian product of the individual variable domains. In the logistics example above, the merging process can explicate the fact that only the values (T, F) and (F, T) are reachable for the merged fluent. Thus it removes the value combinations (T, T) and (F, F) from consideration. The ensuing domain reduction as well as the compilation of negative interactions turns out to be quite useful, as we shall see below.

Notice that when we merge fluents, we naturally wind up with super-fluents that are multi-valued (even if we started with boolean fluents). The popular idea of converting a domain from boolean to multi-valued representation, as described by (Edelkamp & Helmert 1999; Fox & Long 1998; Gerevini & Schubert 2000; Helmert 2006) can thus be seen as just a special case of fluent merging. Specifically these methods focus on merging fluents that have strict mutual exclusion relationships. Fluent merging does not however have to be confined to fluents with strict mutual exclusions. We shall see that merging fluents with strong inter-causal dependencies can also be advantageous.

The most important aspect of fluent merging is that it "compiles-in" some of the negative interactions between the fluents. This is illustrated well in the logistics example above. The negative interaction between $at(truck1, loc1) = T$ and $at(truck1, loc2) = T$ is effectively removed by the fact that the merged fluent doesn't have $\{T, T\}$ in its domain. This has a significant impact on two modern ideas for speeding up planning:[1]

- *Relaxed reachability heuristics* which do reachability analysis by ignoring negative interactions can do a more informed job of distance estimation after fluent merging (since some of the negative interactions have already been compiled-in).
- *Factored planning* techniques that attempt to find plans for individual fluents and combine them can benefit if fluents with dependencies are merged up front (since this means that the merging phase will

[1] While we focus on the compilation of negative interactions, it is worth noting that the earlier work by Edelkamp et. al. (Edelkamp & Helmert 1999) motivates fluent merging from the perspective of minimizing state encoding length.

likely have fewer backtracks).

Both these advantages have been amply, if indirectly, established in the planning literature. Part of the reason for the improved performance of Fast-Downward planner (Helmert 2006) can be attributed to the fact that it does fluent merging (in converting boolean fluent domain description into multi-valued fluent description). Our own recent work (van den Briel *et al.* 2007) shows that this type of informedness advantage also holds for more general forms of fluent merging. The advantage of fluent merging for factored planning is established by our work (van den Briel, Vossen, & Kambhampati 2005) which shows that multi-valued representations can lead to significant performance improvements.

While the foregoing paints a mostly positive picture of fluent merging, as exhorted at the outset, fluent merging can only be good in scenarios where the original domain description contains a significant number of strongly dependent fluents. Specifically, while merging reduces the number of fluents, it increases their domain sizes. The latter increase can be exponential in the number of variables merged. This worst case occurs when the merged variables are completely independent. However, the increase can be offset with domain reduction if the variables are strongly dependent.[2]

The problem with existing fluent merging methods however is not so much that they don't lead to computational advantages, but rather that they are *too conservative*. In particular, the mutual-exclusion based multi-valued models found by (Edelkamp & Helmert 1999; Helmert 2006) consider merging fluents only when the effective domain size goes from exponential to linear. Specifically, they will merge m boolean fluents that form a mutex clique into a single multi-valued variable with m values (which is a reduction of domain size from 2^m to m). While these merging strategies will give considerable computational advantages *when they are applicable*, they are too conservative and are often *not applicable*. Specifically, we may have sets of boolean fluents that have strong dependencies and yet do not quite form a mutex clique. Finding and merging such sets of fluents could still be quite useful. The domain reduction in such cases may only be from 2^m to m^k (for some small k) instead of m–and yet it is impressive nonetheless. For example, our recent work (van den Briel *et al.* 2007) shows that more aggressive merging can improve heuristic informedness. The challenge of course is to come up with approaches that can identify such fluent sets automatically.

[2]It is even possible to have domains where merging *all* the fluents can still be a good idea. Consider the extreme example with n atoms, two legal states $(T, ..., T)$ and $(F, ..., F)$, one action that toggles all variables from T to F, and one action that toggles all variables from F to T . In this case, we are better off merging *all* fluents into a single super-fluent that describes the complete reachable state space of the problem.

In the remainder of the paper, we provide some first steps towards formally defining the fluent merging problem and developing methods that are more aggressive in identifying mergeable fluents. Towards the latter, we describe some techniques based on novel analysis of the causal graph. In a way, our work can be seen as an applied approach to the work on factored planning by Brafman and Domshlak 2006.

This paper is organized as follows. First, we provide some background and define the process of fluent merging more formally. Second, we discuss the potential use of fluent merging in improving heuristic estimates and factored planning. Some conclusions are given at the end.

Fluent Merging

We assume that we are given a SAS+ planning task $\Pi = \langle C, A, s_0, s_* \rangle$, which allows both boolean and multi-valued state descriptions, where:

- $C = \{c_1, ..., c_n\}$ is a finite set of state variables, where each state variable $c \in C$ has an associated domain V_c and an implicitly defined extended domain $V_c^+ = V_c \cup \{u\}$, where u denotes the *undefined value*. For each state variable $c \in C$, $s[c]$ denotes the value of c in state s. The value of c is said to be *defined* in state s if and only if $s[c] \neq u$. The total state space $S = V_{c_1} \times ... \times V_{c_n}$ and the partial state space $S^+ = V_{c_1}^+ \times ... \times V_{c_n}^+$ are implicitly defined.
- A is a finite set of actions of the form $\langle pre, post, prev \rangle$, where *pre* denotes the pre-conditions, *post* denotes the post-conditions, and *prev* denotes the prevail-conditions. For each action $a \in A$, $pre[c], post[c]$ and $prev[c]$ denotes the respective conditions on state variable c. The following two restrictions are imposed on all actions: (1) Once the value of a state variable is defined, it can never become undefined. Hence, for all $c \in C$, if $pre[c] \neq u$ then $pre[c] \neq post[c] \neq u$; (2) A prevail- and post-condition of an action can never define a value on the same state variable. Hence, for all $c \in C$, either $post[c] = u$ or $prev[c] = u$ or both. We use A_c^E to denote the actions that have an effect in state variable c, and A_c^V to denote the actions that have a prevail condition in c.
- $s_0 \in S$ denotes the initial state and $s_* \in S^+$ denotes the goal state. We say that state s is *satisfied* by state t if and only if for all $c \in C$ we have $s[c] = u$ or $s[c] = t[c]$. This implies that if $s_*[c] = u$ for state variable c, then any defined value $f \in V_c$ satisfies the goal for c.

Two important constructs that we use are the so-called *domain transition graph* and *causal graph*. The domain transition graph $DTG_c = (V_c, E_c)$ of state variable c is a labeled directed graph with nodes for each value $f \in V_c$. DTG_c contains a labeled arc $(f, g) \in E_c$ if and only if there exists an action a with $pre[c] = f$ and $post[c] = g$ or $pre[c] = u$ and $post[c] = g$. Each arc is labeled by the set of actions with corresponding pre-

and post-conditions. For each arc (f_1, f_2) with label a in DTG_c we say that there is a *transition* from f_1 to f_2 and that action a has an *effect* in c. The causal graph $CG_\Pi = (V, E)$ of a planning task Π is a directed graph with nodes for each state variable $c \in C$. CG contains an arc $(c_1, c_2) \in E$ if and only if there exists an action a that has a prevail condition or precondition in c_1 and an effect in c_2.

We define fluent merging as the *composition* of two or more state variables as follows. The term composition is also used in model checking to define the parallel composition of automata (Cassandras & Lafortune 1999).

Definition *(Composition) Given the domain transition graph of two state variables c_1, c_2, the composition of DTG_{c_1} and DTG_{c_2} is the domain transition graph $DTG_{c_1||c_2} = (V_{c_1||c_2}, E_{c_1||c_2})$ where*

- $V_{c_1||c_2} = V_{c_1} \times V_{c_2}$
- $((f_1, f_2), (g_1, g_2)) \in E_{c_1||c_2}$ *if $f_1, g_1 \in V_{c_1}$, $f_2, g_2 \in V_{c_2}$ and there exists an action $a \in A$ such that one of the following conditions hold.*
  - $pre[c_1] = f_1$, $post[c_1] = g_1$, *and* $pre[c_2] = f_2$, $post[c_2] = g_2$
  - $pre[c_1] = f_1$, $post[c_1] = g_1$, *and* $prev[c_2] = f_2$, $f_2 = g_2$
  - $pre[c_1] = f_1$, $post[c_1] = g_1$, *and* $f_2 = g_2$

We say that $DTG_{c_1||c_2}$ is the composed domain transition graph of DTG_{c_1} and DTG_{c_2}.

Example Consider the set of actions $A = \{a, b, c, d\}$ and the set of state variables $C = \{c_1, c_2\}$ whose domain transition graphs have $V_{c_1} = \{f_1, f_2, f_3\}$, $V_{c_2} = \{g_1, g_2\}$ as the possible values, and $E_{c_1} = \{(f_1, f_3), (f_3, f_2), (f_2, f_1)\}$, $E_{c_2} = \{(g_1, g_2), (g_2, g_1)\}$ as the possible transitions as shown in Figure 1. Merging state variables c_1 and c_2 creates a new state variable whose domain is defined by the Cartesian product $V_{c_1} \times V_{c_2}$ as shown in Figure 1. Note that some value combinations become disconnected components, such as (f_3, g_2). These disconnected components are unreachable from the initial state and thus can safely be ignored. Also, note that some actions generate multiple instances in the composition, such as actions c and d. These multiple instances are generated if an action has an effect in one fluent, but no effect or prevail condition in the other fluent[3].

The composition of more than two state variables can be obtained by creating a composition over one or more composed domain transition graphs. For example, $DTG_{c_1||c_2||c_3}$ can be obtained by creating the composition between $DTG_{c_1||c_2}$ and DTG_{c_3}.

[3]As we shall see later, one consideration in picking effective merging strategies is to ensure that they don't increase the number of actions too much.

Identifying Mergeable Fluents

Previously, mergeable fluents have been identified by looking at the boolean fluents that form a mutex clique. These type of fluent mergings are good since they eliminate many unreachable value combinations. We introduce two other ways to identify mergeable fluents based on causal graph analysis.

First, in order to identify mergeable fluents we look for cycles in the causal graph. Causal cycles are undesirable as they describe two-way dependencies between state variables. That is, changes in state variable c_1 will depend on conditions in state variable c_2, and vice versa. While it is possible that causal cycles involve more than two state variables, we only consider 2-cycles (cycles of length two). In particular, we merge two fluents c_1 and c_2 if they form a 2-cycle in the causal graph and if the following condition hold.

- For all $a \in A^E_{c_1}$ we have $a \in (A^E_{c_2} \cup A^V_{c_2})$
- For all $a \in A^E_{c_2}$ we have $a \in (A^E_{c_1} \cup A^V_{c_1})$

In other words, for every action a that has an effect in state variable c_1 (c_2) we have that action a has an effect or prevail condition in state variable c_2 (c_1). The main reason for requiring this additional condition is to ensure that the actions do not generate multiple instances in the composition. This condition is quite restrictive, but as shown by the next example effective nevertheless. Moreover, van den Briel *et al.* 2007 show that this type of fluent merging leads to improved network flow based reachability heuristics.

Example Figure 2 shows an example of how fluent merging can remove causal 2-cycles from the causal graph. The figure on the left shows the causal graph for a typical state description of a Zenotravel problem with two airplanes, two passengers, and any number of cities. The state description is determined by six state variables: one for each passenger $Loc(person1)$ and $Loc(person2)$ with values that denote the location of the passengers, one for each airplane $Loc(airplane1)$ and $Loc(airplane2)$ with values that denote the location of the airplanes, and one for the fuel tank of each airplane $Fuellevel(airplane1)$ and $Fuellevel(airplane2)$. The figure on the right shows the causal graph of the same problem, but is based on a state description in which the state variables $Loc(airplane1)$ and $Fuellevel(airplane1)$, and $Loc(airplane2)$ and $Fuellevel(airplane2)$ have been merged into super state variables. The advantageous of the resulting state description should be clear. Fewer cycles in the causal graph will lead to better hierarchical decompositions, which could lead to improved planning performance.

Second, in order to identify mergeable fluents we look at pairs of atoms (f_1, f_2) such that there exists an action that has f_1 as a prevail condition and f_2 as a delete effect. Specifically, we look for causal links in the causal graph that are introduced by the actions with a prevail condition in one fluent and an effect in another flu-

ent. Some hierarchical based planners can handle such causalities quite well and simply incorporate them directly into the hierarchical structure.

For example, in the Logitics domain a hierarchical planner may first find a plan for each package, use these plans to impose ordered conditions on the trucks, and then find a plan for each truck. However, reachability heuristics that do not exploit hierarchies may sometimes give poor estimates even in some very simple planning tasks.

Example Figure 3 shows an example of how fluent merging can improve heuristic estimates. The figure considers a simple Logistics problem with one truck, one package, and two locations. In the initial state we have the truck at 2 ($= at(truck1, loc2)$) and the package at 1 ($= at(package1, loc1)$). Several known reachability heuristics, including FF's relaxed plan heuristic (Hoffmann & Nebel 2001), fail to recognize that the truck needs to drive back to location 2 in order to unload the package. The figure shows the merged atom pairs and their corresponding transitions. If FF's relaxed plan heuristic considers the atom pairs as single atoms, it would have detected that it needs to drive to location 1 to load the package and then drive back to unload the package.

Conclusions

We described the process of fluent merging and showed how it can help improve reachability heuristics and factored planning. While fluent merging has been around under the idea of converting boolean to multi-valued representations, we introduced methods that are more general in identifying mergeable fluents.

Our recent work (van den Briel *et al.* 2007) shows that we can derive more informed heuristics by merging fluents without experiencing too much computational overhead. We believe, however, that there may be other ways to identify mergeable fluents, which either extend or generalize the ways that we described.

References

Brafman, R., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006)*, 809–814.

Cassandras, C., and Lafortune, S. 1999. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the European Conference on Planning (ECP-1999)*, 135–147.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:367–421.

Gerevini, A., and Schubert, L. K. 2000. Discovering state constraints in DISCOPLAN: Some new results. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, 761–767.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proceedings of the International Conference of Principles and Practice of Constraint Programming (CP-2007)*. (To appear).

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2005)*, 161–170.

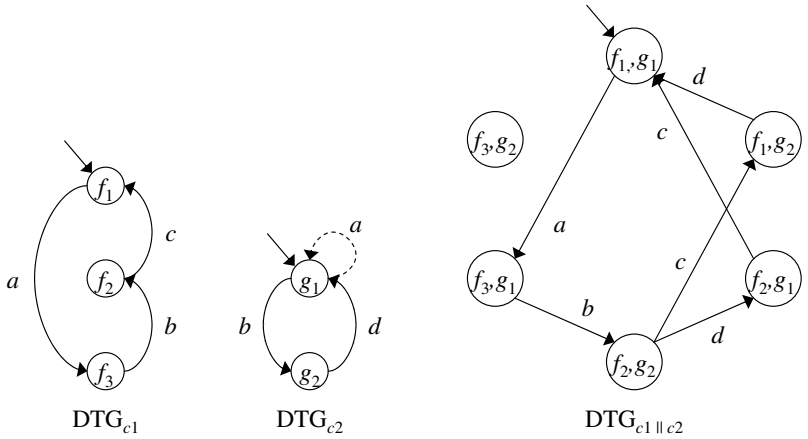

Figure 1: Two domain transition graphs and their composition. Small in-arcs denote the initial state of each state variable.

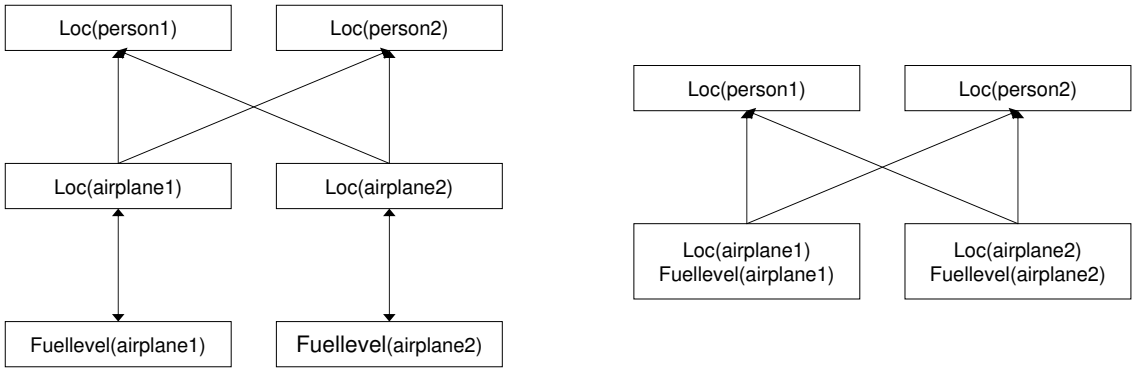

Figure 2: Fluent merging removes causal 2-cycles from the causal graph for a typical state description of the Zenotravel domain.

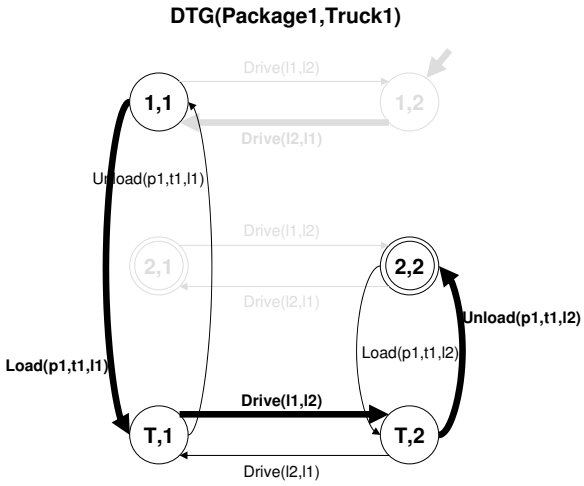

Figure 3: Fluent merging improves heuristic estimates in the Logistics domain.