# Exploiting Search Space Structure in Classical Planning: Analyses and Algorithms (Dissertation Abstract)

**Masataro Asai**
Graduate School of Arts and Sciences
University of Tokyo

## State of the Current Work, Future Plans and Expectations from the Consortium

The author has completed 2 years of research in Masters degree and is in the first year of the PhD, which is not so close to the dissertation. As a result, this dissertation abstract contains several speculative materials. This is because the author's current publications lack the coherent story, primarily due to the lack of good understanding of macro operators and search algorithms in the planning community. I address this issue in the future work sections and make up the coherent story that is necessary to form a viable thesis.

At the time of writing this, I expect from the Consortium the advice how to form a viable, coherent dissertation thesis, which is completely different from writing an individual research paper. I also wish to connect with mentors and students in the Consortium for future collaboration, because some of future work may not make way into the thesis.

In the following sections, I first present my past work, then I propose some future ideas.

## Current Work

### Factored Planning System CAP

We proposed a Factored Planning framework CAP (Asai and Fukunaga 2015).

Factored Planning (FP) is a class of planning framework which first decompose a problem into (hierarchical) subproblems, then (hierarchically) merge the results of the subproblems into a concrete solution of the entire problem. FP subsumes Hierarchical Task Network in which the decomposition is written by humans. In contrast, recent FP systems use the automatic decomposition of the planning problems (Amir and Engelhardt 2003; Brafman and Domshlak 2006; Kelareva et al. 2007; Fabre et al. 2010).

CAP is a variant of FP systems which only weakly requires the decomposability of the problem. Previous FP systems assume the full *disjointness* (subgoals do not conflict with each other) and the *concatenability* (high-level solver can connect the solutions of the decomposed subproblems), primarily because it tries to solve the problem using *all and only* the solutions to the decomposed subproblems. CAP, in contrast, uses the solutions to the subproblems as macro operators, and compose the plan using macros as well as the primitive actions.

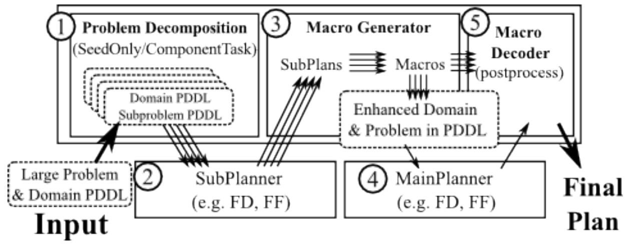

Figure 1: CAP system overview. SubPlanner and MainPlanner are domain-independent planners, e.g., FD/lama (Helmert 2006), FF (Hoffmann and Nebel 2001). They can be the same planner, or different planners.

Figure 1 shows the overview of the CAP framework. SubPlanner and MainPlanner are domain-independent planners, e.g., FD/lama (Helmert 2006), FF (Hoffmann and Nebel 2001), Probe (Lipovetzky and Geffner 2011), YAHSP3 (Vidal and others 2004; Vidal 2011; 2014). They can be the same planner, or different planners (mixed configuration). In detail, CAP works as follows:

1. *Problem Decomposition*: Perform a static analysis of the PDDL problem in order to identify the independent subproblems. Each subproblem is called a *component task*, which is created from an *abstract component*. There are several ways to construct abstract components, which affect the resulting component task.
2. *Generate Subplans with SubPlanner*: Solve the subproblems with a domain-independent planner (SubPlanner).
3. *Macro generation*: For each subplan, concatenate all of its actions into a single, ground (nullary) macro operator.
4. *Main Search by MainPlanner*: Solve the augmented PDDL domain (including macros) with a standard domain-independent planner (MainPlanner).
5. *Decoding*: Finally, any macros in the plan found by MainPlanner are decoded back to the primitive actions.

Unlike the previous Factored Planning frameworks, CAP was shown to be capable of solving wide range of planning problems. We tested CAP in extremely large planning problems generated by the same problem generators in the standard IPC Sequential Satisficing domains, as well as the Learning Track Test instances of IPC. Table 1 shows that CAP and MUM, a state-of-the-art macro learning system, improve performance in a completely different domains, and that CAP combined with MUM further improves the performance.

| | | $X$ = FF | | | | $X$ = FD/lama | | | | $X$ = Probe | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Domain | FF | MUM(FF) | $\text{CAP}_s^{7.5}$(FF) | MUM($\text{CAP}_s^{7.5}$(FF)) | FD/lama | MUM(FD/lama) | $\text{CAP}_s^{7.5}$(FD/lama) | MUM($\text{CAP}_s^{7.5}$(FD/lama)) | Probe | MUM(Probe) | $\text{CAP}_s^{7.5}$(Probe) | MUM($\text{CAP}_s^{7.5}$(Probe)) |
| **IPC2011 Learning** | barman-ipc11-learn(30) | 0 | 0 | 29 | **30** | 5 | 0 | **29** | 0 | 9 | 1 | 24 | **30** |
| | blocksworld-ipc11-learn(30) | 6 | **25** | 6 | **25** | 25 | **29** | 25 | **29** | 19 | **29** | 20 | **29** |
| | depots-ipc11-learn(30) | 2 | **3** | 1 | 1 | 0 | 0 | 0 | 0 | 28 | 29 | 27 | **30** |
| | gripper-ipc11-learn(30) | 0 | 0 | 0 | 0 | 0 | **5** | 0 | **5** | 0 | **30** | 0 | **30** |
| | parking-ipc11-learn(30) | **1** | **1** | **1** | **1** | **14** | **14** | 8 | 10 | **4** | 2 | 3 | 2 |
| | rover-ipc11-learn(30) | 2 | 0 | 3 | **4** | **27** | 0 | 12 | 23 | 15 | 0 | 10 | **19** |
| | satellite-ipc11-learn(30) | 2 | 1 | 2 | **3** | **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | spanner-ipc11-learn(30) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | tpp-ipc11-learn(30) | 0 | 9 | 20 | **30** | 14 | **30** | **30** | **30** | **10** | 0 | **10** | 0 |
| | Sum | 13 | 39 | 62 | **94** | 90 | 78 | **104** | 97 | 85 | 91 | 94 | **140** |

Table 1: IPC2011 Learning Track results on 15 minutes, 4GB memory setting, using the standard planner $X \in \{\text{FD/lama, FF, Probe}\}$, with/without either/both of macros introduced by CAP and MUM.

CAP has a plenty of rooms for enhancements. It can be enhanced by using the different planners in the subproblem solving and the main planning enhanced by macros. The timelimit criteria of the subproblem solving can be dynamically optimized by the iterative resource allocation. Some subproblems can be pruned by the compatibility criteria between the subproblems, which is checked by detecting the graph isomorphism. CAP can also be enhanced with a "restoration macro", a macro that "bridges the gap" to the next applicable macro.

## Revisiting the Utility Problem: An Empirical Analysis

Although the performance improvement of CAP is clear, we gave further in-depth analysis on why CAP system works and why their enhancements work.

With this task in mind, we revisited the *Utility Problem*, a tradeoff between the benefit and the cost of introducing macros. Although recent macro systems such as MacroFF (Botea et al. 2005), Wizard (Newton et al. 2007) and MUM (Chrpa, Vallati, and McCluskey 2014) employ sophisticated macro pruning methods, some of key assumptions regarding the utility problem predate current heuristic search based planners. We reinvestigate the utility problem for macro operators using two models, "partial solution macros" and "junk macros", each represents how "obviously useful" macros and "obviously useless" macros affect the search performance of planners. As a result, we get the following observations:

First, contrary to conventional wisdom, macro operators do not increase the *effective branching factor* in modern heuristic search-based planners. We show that introducing randomly chosen "junk" macros reduces node evaluations in many domains, and in some domains, junk macros improves the runtime (Table 2).

| (LAMA) Domain | L | Preprocess [sec] | Search [sec] | Total [sec] | Eval [node] |
|---|---|---|---|---|---|
| airport | 8 | *112 (1.1)* | **355 (.50)** | **467 (.57)** | **280721 (.74)** |
| cybersec | 8 | **2217 (.91)** | 3 | **2220 (.91)** | 3309 |
| depot | 8 | 22 (1.3) | 149 (.50) | **171 (.54)** | **190577 (.47)** |
| driverlog | 5 | *24 (1.3)* | *105 (1.6)* | *129 (1.5)* | **179752 (.88)** |
| hanoi | 2 | 3 (1.0) | **287 (.79)** | **290 (.79)** | 2070986 (.97) |
| mystery | 5 | *87 (1.4)* | **4 (.21)** | *91 (1.1)* | **2643 (.08)** |
| pipesworld-t | 8 | *304 (1.5)* | *893 (2.1)* | *1197 (1.9)* | **355576 (.89)** |
| rovers | 2 | *331 (1.1)* | 114 (.96) | *445 (1.0)* | **87475 (.90)** |
| transport-sat11 | 2 | *205 (1.3)* | *630 (2.0)* | *835 (1.8)* | **47244 (.47)** |

Table 2: Selective results showing the improvements by junk macros of length $L$, using LAMA planner. Each cell shows the sum over all instances in the domain solved by all configurations, averaged by the 10 runs. Ratios relative to LAMA are shown, e.g., "(.86)" means the ratio compared to LAMA is 0.86. **Improvement**/*degradation* are tested with statistical significance ($p < 0.001$).

Next, we show that the planner may fail to use even trivially useful "partial solution macros".

The most trivially useful macros are the complete solutions to the planning problem itself — Any solution can be encoded as a macro, such that applying it to the initial state results in reaching the goal in one step. Although such macros are clearly unrealistic, understanding the behavior of modern planners with such a macro can yield useful insights.

As a next step we investigate *partial solution macros*, which are the macros generated by splitting a solution into several parts and encoding the individual pieces as macros. Since connecting those macros solve the entire problem instantly, smart planners *should* be able to successfully connect them. We refer to this assumption a *concatenability* assumption, an important assumption made by Factored Planners. However, we empirically show that the planners are in fact not able to connect them, and the concatenability assumption does not hold. We show that an important factor determining such success/failure in utilizing macros is the difficulty of establishing a chain of macro applications, i.e., the "gap" between the partial solution macros (Table 3).

By applying new insights, we can now fully investigate CAP and *restoration macros*, an enhancement to CAP which addresses the problem of large gaps between the macros found by CAP.

| | cov. | macros | ($L \geq 2$) | usage | (%) | expansion | time |
|---|---|---|---|---|---|---|---|
| baseline | 557 | 0 | 0 | 0 | 0/0 | 83009511 | 1765 |
| split1 | 561 | 598 | 595 | 557 | 93.6 | 16194 | 0.36 |
| split3 | 561 | 1794 | 1727 | 1550 | 89.7 | 175689 | 3.74 |
| split10 | 561 | 5980 | 4100 | 2999 | 73.1 | 3683892 | 50.2 |
| split3gap1 | 561 | 1794 | 1648 | 1423 | 86.3 | 389398 | 20.6 |
| split3gap3 | 560 | 1794 | 1444 | 1158 | 80.2 | 1811416 | 74.7 |
| split3gap5 | 561 | 1794 | 1260 | 984 | 78.1 | 7540669 | 202 |

Table 3: Results on problems with partial solution macros and partial solution macros with gaps.

## Tiebreaking Strategy for A*: How to Explore the Final Frontier

Despite recent improvements in search techniques for cost-optimal classical planning, the exponential growth of the size of the search frontier in A* is unavoidable. We investigate tiebreaking strategies for A*, experimentally analyzing the performance of standard tiebreaking strategies that break ties according to the heuristic value of the nodes. We find that tiebreaking has a significant impact on search algorithm performance when there are zero-cost operators that induce large plateau regions in the search space. We develop a new framework for tiebreaking based on a depth metric which measures distance from the entrance to the plateau, and proposed a new, randomized strategy which significantly outperforms standard strategies on domains with zero-cost actions (Asai and Fukunaga 2016).

We showed that contrary to conventional wisdom, tiebreaking based on the heuristic value is not necessary to achieve good performance. We also proposed a new framework for defining tiebreaking policies based on *depth*. Our depth-based, randomized strategy $[h, rd, ro]$, which uses the heuristic value, but explicitly avoids depth and ordering biases present in previous methods, significantly outperforms previous strategies on domains with zero-cost actions, including practical application domains with resource optimization objectives in the IPC benchmarks. The proposed approach is highly effective on domains where zero-cost actions create large plateau regions where all nodes have the same $f$ and $h$ costs and the heuristic function provides no useful guidance.

## Summary of Contributions

Our current contributions can be summarized as follows. (1) We proposed CAP, a satisficing factored planner using macros. (2) We investigated of the general effect of macro operators in satisficing planning, and applied the new observation to CAP. (3) We investigated the past tiebreaking strategies of A* for optimal search, and proposed a new tiebreaking methods which diversifies the search depth.

Although (1) and (2) are the same line of work, (3) does not nicely fit into the storyline, which will be fixed in the future work as proposed in the following sections.

## Introduction (Future Work)

Current State-of-the-Art planner such as Fast Downward (**?**) can solve the planning problems of a moderately large size in a reasonable amount of time, mainly thanks to the greedy forward search combined with sophisticated heuristic functions such as *delete-relaxation* (**?**) and *landmarks* (Richter and Westphal 2010), combined with techniques specifically tailored toward planning problems such as *helpful actions* (**?**) and *consistency criteria* (Lipovetzky and Geffner 2011).

However, Classical Planning is PSPACE-Complete (Bylander 1994) and intractable in general. Above strategies are made upon the assumption that the problems are serially decomposable, and in fact its usefullness does not hold in the random problem instances generated by algorithm A, B or C in (Bylander 1996; Rintanen and others 2004) nor in some domains such as Floortile, Scanalyzer in recent IPCs (Alcázar, Veloso, and Borrajo 2011).

Moreover, there are several satisficing search strategies that seem still yet relatively incompatible to, or independent from the heuristic forward search. Examples include SAT-based planners (Rintanen 2012), Lookaheads (Vidal and others 2004), Macro actions (Chrpa, Vallati, and McCluskey 2015), Factored Planning (Amir and Engelhardt 2003; Asai and Fukunaga 2015), Diversified Search (Imai and Kishimoto 2011; Xie et al. 2014; Burfoot, Pineau, and Dudek 2006).

In our work, we try to provide a consistent theoretical background unifying all these strategies, and then propose several practical algorithms inspired by the new observations.

## Macro-conversion of the Search Algorithms

First, we formally define the notion of *best first search with lookaheads (L-BFS)* and show that macro actions can simulate any L-BFS, and vice versa (L-BFS can simulate any macro actions). The intuition is as follows: When BFS starts a depth-first lookahead during the search in a certain condition, that condition can be directly encoded in the preconditions of the macros, although in a problem-specific manner. This unifies various inadmissible search strategies as a modification of the search space using macro actions, which greatly simplifies the discussions in the later sections. We hereafter call the act of simulating L-BFS by macro operators as "macro-conversion".

## Phase Transition of the Search Space

Next, we tackles the problem of *Phase Transition* in the complex search space of planning problems. Phase transition in a class of search problems is a phenomenon that the difficulty and the complexity of the problems are ruled by a simple meta-level parameter, and become increasingly easy or hard when the parameter crosses a *critical value*.

In AI research, phase transition was first found in the boolean satisficing problems (Huberman and Hogg 1987; Cheeseman, Kanefsky, and Taylor 1991; Selman, Mitchell, and Levesque 1996) and are recently connected to the physical phenomenon in the Ising model of the spin grass (Barahona 1982). In boolean-SAT problems, the meta-level parameter is the ratio $r = L/N$ of the number of clauses $L$ and the number of propositions $N$, with a critical value $r_c \approx 4.24$ (Crawford and Auton 1993). In boolean SAT, when $N \to \infty$, the probability of being SAT is 0 when $r < r_c$ and 1 when $r > r_c$. When $N$ is finite, it becomes increasingly difficult to determine the satisfiability when $p$ approaches $p_c$ from either above or below.

In planning problems, previous strategies for analyzing the phase transition are primarily based on the analogy from the boolean satisficing problems. For example, the meta-parameter that is claimed to be controlling the problem difficulty is the ratio of number of operators versus the number of state variables (Rintanen and others 2004). In Algorithm B, (Bylander 1996) A and C (Rintanen and others 2004), planning operators are generated randomly.

We instead analyze the planning problems based on the *Percolation Theory* (Stauffer and Aharony 1994), a theory describing the behavior of the fluid percolating through porous material from one end to the other end. The same theory is already shown to be applied to the pathfinding on random graph and ACO algorithm (Velloso and Roisemberg 2008) because the existence of a satisficing path in a graph is equivalent to percolating the material from one porous site to the goal site with the fluid. However, the search spaces of planning problems and the random graphs are claimed to have the different characteristics (Bylander 1996; Rintanen and others 2004).

Percolation theory dictates that the connectability of the graph is controlled simply by the ratio $p$ of the number of *occupied edges* to the number of all edges. In the infinite graph, the probability $p$ of two points having a path is 0 when the ratio $r$ is below a critical threshold $r_c$, and is 1 when $r > r_c$. The value of $r_c$ depends on the topology of the graph. In case of finite graph, the probability $p$ becomes a continuous function $p(r)$ which has a *critical region* around $r_c$ where the value grows from 0 to 1. The width of the region is called *correlation index radius*, which basically means the radius in which a node is affected by the other nodes.

Using these theories, we treat the grounded search space directly, rather than through the number of operators. An operator does not represent a single edge in the search space, and instead they representing multiple edges starting from the states which satisfies the precondition — the partial specification of the states. We plan to propose a new random problem generation methods which considers the number of states that is applicable to each operator, and show it achieves a much steeper phase transitions than the previous methods. We will also provide a formal proof that the SAT/UNSAT of the problem approaches to 0/1 around the critical value as the size of the graph approaches to the infinity.

## Restart-based, Probabilistically Complete Search Algorithm with Randomly Reduced Number of Edges

We propose a restart-based search algorithms which solve the problems by randomly removing the edges in the search space. The reduced instances may be UNSAT, but we show that as long as we control the number of edges so that the meta-parameter $r$ is above the critical region, we can still solve the problems asymptotically as we restart with different random seeds.

This method has an effect of shifting the meta-level parameter outside the critical region and making the problem easily SAT/UNSAT, which follows the intuitive observation that the search finishes quickly due to the reduced branching factor, or the problem is quickly proven to be UNSAT using reachability analysis on the relaxed planning graph. We plan to empirically show that this method achieves a good performance in IPC problems.

## Extensions of Tiebreaking Strategy for A* to Satisficing Planning

We analyse our tiebreaking strategy for A* (Asai and Fukunaga 2016) using macro-conversion and percolation theory. Since the strategy explores the search space sparsely, it would have the similar effect as the previous algorithm (Search Algorithm with Randomly Reduced Number of Edges) on the plateau region of A*.

Using the same tiebreaking strategy, we also propose a constant-error search method as compared to the famous constant-*times*-error method WA*. It divides f-value by a constant error value $c$, ignoring the remainder. Since it introduces an intensive increase of the plateau region, we use the same tiebreaking strategy as in (Asai and Fukunaga 2016).

Another possible application of this tiebreaking is the temporal planning problems, where the actions with short duration can be hidden behind the actions with longer duration, which is known as $\epsilon$-cost traps (Cushing, Benton, and Kambhampati 2010).

## Analysing CAP using Percolation Thoery

Finally, we analyse CAP using Percolation Theory. Since the macros introduced by CAP tends to be long, it has a significant impact on the connectability of the search space. This analysis is expected to finally form a into a coherent story out of the current work which have the different topics.

## Conclusion

I summarized several current work of mine (including the materials being under review) and showed that they have diverged topics which are hard to form a coherent thesis. Then I proposed an idea how to merge those topics into a single topic, percolation theory, using the macro-conversion technique.

## References

Alcázar, V.; Veloso, M.; and Borrajo, D. 2011. Adapting a rapidly-exploring random tree for automated planning. In *SoCS*.

Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI*, volume 3, 929–935. Citeseer.

Asai, M., and Fukunaga, A. 2015. Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*.

Asai, M., and Fukunaga, A. 2016. Tiebreaking Strategies for Classical Planning Using $A^*$ Search.

Barahona, F. 1982. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General* 15(10):3241.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *JAIR* 24:581–621.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when, and when not. In *AAAI*, volume 6, 809–814.

Burfoot, D.; Pineau, J.; and Dudek, G. 2006. RRT-Plan: A Randomized Algorithm for STRIPS Planning. In *Proceedings* *of the International Conference of Automated Planning and Scheduling(ICAPS)*, 362–365.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69(1):165–204.

Bylander, T. 1996. A Probabilistic Analysis of Prepositional STRIPS Planning. *Artificial Intelligence* 81(1):241–271.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *IJCAI*, volume 91, 331–340.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use. In *ICAPS*, 65–73.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2015. On the Online Generation of Effective Macro-Operators. In *International Joint Conference on Artificial Intelligence*, 1544–1550.

Crawford, J. M., and Auton, L. D. 1993. Experimental results on the crossover point in satisfiability problems. In *AAAI*, volume 93, 21–27. Citeseer.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost Based Search Considered Harmful.

Fabre, E.; Jezequel, L.; Haslum, P.; and Thiébaux, S. 2010. Cost-Optimal Factored Planning: Promises and Pitfalls. In *ICAPS*, 65–72.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *JAIR* 14:253–302.

Huberman, B. A., and Hogg, T. 1987. Phase transitions in artificial intelligence systems. *Artificial Intelligence* 33(2):155–171.

Imai, T., and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *AAAI*.

Kelareva, E.; Buffet, O.; Huang, J.; and Thiébaux, S. 2007. Factored Planning Using Decomposition Trees. In *IJCAI*, 1942–1947.

Lipovetzky, N., and Geffner, H. 2011. Searching for Plans with Carefully Designed Probes. In *ICAPS*.

Newton, M. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, 256–263.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.(JAIR)* 39(1):127–177.

Rintanen, J., et al. 2004. Phase Transitions in Classical Planning: An Experimental Study. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, volume 2004, 101–110.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Selman, B.; Mitchell, D. G.; and Levesque, H. J. 1996. Generating hard satisfiability problems. *Artificial intelligence* 81(1):17–29.

Stauffer, D., and Aharony, A. 1994. *Introduction to percolation theory*. CRC press.

Velloso, B. P., and Roisemberg, M. 2008. Percolation analyses in a swarm based algorithm for shortest-path finding. In *Proceedings of the 2008 ACM symposium on Applied computing*, 1861–1865. ACM.

Vidal, V., et al. 2004. A Lookahead Strategy for Heuristic Search Planning. In *Proceedings of the International Conference of Automated Planning and Scheduling(ICAPS)*, 150–160.

Vidal, V. 2011. YAHSP2: Keep it simple, stupid. *Angel Garcıa-Olaya, SJ, and López, CL, eds., Seventh International Planning Competition* 83–90.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. *Vallati, Mauro and Chrpa, Lukáš and McCluskey, Thomas L., Eighth International Planning Competition* 64–65.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2395–2402.