Evaluation of a Simple, Window-based, Replanning Approach to Plan Optimization

Shoma Endo, Masataro Asai, Alex Fukunaga
Graduate School of Arts and Sciences
The University of Tokyo

Abstract

The task of satisficing planning is to find the highest quality plan within a given time budget. Several postprocessing techniques for plan optimization techniques have been developed for improving the quality of a satisficing plan. In this paper, we first survey a class of plan optimization techniques which apply replanning to windows (subplans) of a given plan. We propose and evaluate a CH-WIN, a new window-based plan optimization algorithm based on AIRS, a previous technique for prioritizing plan windows according to discrepancies between actual and heuristic cost accruals within a window.

1 Introduction

Minimizing the cost of the solution is a key element in satisficing planning in order to obtain a reasonable solutions for practical real-world applications. There are two major approaches to addressing this problem. The first approach is the development of anytime search algorithms that initially generate some solution relatively quickly, then continue searching for better solutions in the remaining time available. Examples of this approach include Restarting WA* (Richter and Westphal 2010), AEES (Thayer, Benton, and Helmert 2012), Diverse Anytime Search (Xie, Valenzano, and Müller 2013) and so forth. Another line of work includes plan optimization techniques, such as Action Elimination (Nakhost and Müller 2010), Action Dependency (Chrpa, McCluskey, and Osborne 2012), Plan Neighborhood Graph Search(PNGS) (Nakhost and Müller 2010), Block Deordering (Chrpa and Siddiqui 2015). Although these approaches share the same objective, the key difference between anytime search algorithms and postprocessing algorithms is that the former can be applied to generate a plan from scratch, whereas the latter assumes that at least one satisficing plan is available as an input.

This paper focuses on plan optimization techniques. In particular, we investigate a class of techniques we call "window-based plan optimization". In this approach, some part of an existing plan (a "window") is selected, and a replanning algorithm is applied to this window in order to locally optimize the selected subplan. This process is repeated until time runs out.

We first evaluate R-WIN, a simple, baseline strategy which randomly selects the windows to be optimized. We then propose CH-WIN, an improved variation of AIRS, a previous method for plan optimization which was evaluated on domain-specific solvers for the 15-puzzle and a grid pathfinding domain AIRS (Estrem and Krebsbach 2012). CH-WIN uses the same criteria as AIRS for ranking candidate replanning windows, but uses an improved method for window selection. We show that CH-WIN outperforms previous algorithms including AIRS and PNGS.

Finally, we evaluate the performance of these optimizers when combined in sequence. Since the input of an optimization algorithm is a plan and its output is also a plan, we can apply a chain of several optimization algorithms. Although this requires additional optimization time, we empirically show that spending more time on optimization actually pays off, compared to the performance of a single plan optimization technique.

This paper is structured as follows. Section 2 reviews several existing plan-optimization frameworks. Section 3 propose a class of window-based plan optimization and its two instances, and evaluate their performance against existing framework. Section 4 investigates the performance of the sequence of several plan-optimization frameworks. We finally conclude with future remarks.

2 Background

A common strategy for plan optimization in sequential satisficing planning is to run an anytime search algorithm with some time limit, then allocate the remaining time to the postprocessing optimizer. The postprocessing phase can be iterated, so that it continues to improve the plan quality.

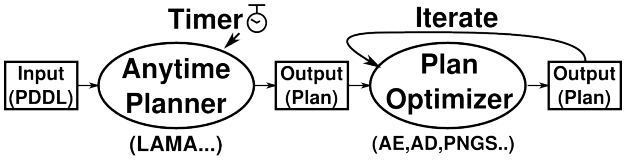

Figure 1: The concept of anytime planning combined with post-processing plan optimization.

Some algorithms run in polynomial-time while the others are based on search algorithms that may consume exponential amount of time. We call the first variants as the *polytime* optimization algorithms while the latter as the *search-based* optimization algorithms.

Action Elimination [**AE**] (Nakhost and Müller 2010) is a polytime optimization algorithm which iterates over the actions in the plan looking for unnecessary actions whose removal does not make the rest of the plan infeasible. A straightforward implementation runs in $O(pn^2)$ where n is the length of the plan and p is the maximum number of preconditions among actions in the plan.

Action Dependency analysis [**AD**](Chrpa, McCluskey, and Osborne 2012) is also a polytime optimization algorithm. It consists of two major steps. The first step uses a notion of *direct dependency* between a pair of actions and its transitive extension called *action dependency*. It removes all actions that is not dependent on the goal action, a pseudo action whose precondition is same as the goal condition. In the second step, AD uses the notion of *inverse action* which represents a pair of actions with effects that cancels each other. It removes a pair of inverse actions from the plan when the actions surrounded by the pair are not affected by the removal.

The first search-based optimization algorithm we are aware of is by Ratner and Pohl, who proposed a plan optimization algorithm which divides a suboptimal plan in to segments of length d, and tries to replace each segment with a better subplan obtained by applying A^* to locally optimize the segment (Ratner and Pohl 1986)

Plan Neighborhood Graph Search [**PNGS**] (Nakhost and Müller 2010) is a search-based plan optimization technique. It takes two parameters A and N, where A is a search algorithm and N is an integer. PNGS generates a *Plan Neighborhood Graph* of the plan as follows: Given a plan π and initial state s_0, consider the plan path $P = s_0, s1, ..., s_g$, which is the sequence of states that the system transitions through when each step of π is applied in sequence. For each such state s_i, it runs A until it generates N nodes. The resulting set of nodes, which includes the original solution itself, is then searched by another algorithm M', which is brute-force Dijkstra's algorithm in their experiments. When the refinement finishes, it doubles the number of nodes N and reapplies the algorithm.

Balyo, Barták, and Surynek investigated a window based, replanning approach for temporal planning which seeks to optimize makespan by applying a SAT-based planner to random or fixed sized windows of a suboptimal plan (Balyo, Barták, and Surynek 2012).

Block-deordering is a recent approach in which a totally ordered plan is first deordered and decomposed into a set of blocks (partially ordered plans), where each block is a set of steps that must not be interleaved with steps that are not in the block. Windows for planning are extracted based on these blocks, and a large-neighborhood search algorithm is applied in order to optimize the subplans (Siddiqui and Haslum 2013; 2015).

Finally, **Anytime Iterative Refinement of a Solution** (Estrem and Krebsbach 2012, AIRS) is a search-based plan optimization technique which runs a local search on a part of the plan. The portion is selected based on *Greedy Plateaus*, i.e. the parts of the plan where the heuristic estimate does not change as much as they should (relative to the actual costs). For every pair of states s_i, s_j in a plan, it calculates the ratio of the heuristic distance $h(s_i, s_j)$ to the actual cost $c(s_i, s_j)$ of the path between two states. In each iteration, the portion of the plan starting from s_i ending at s_j with the minimum ratio $\frac{h(s_i,s_j)}{c(s_i,s_j)-c_{min}}$ is selected, where c_{min} is the smallest action cost in the domain, and start the local search from s_i to s_j, using an admissible bidirectional search. It also has a mechanism to avoid searching the same region many times by storing the information about the portion where the admissible search failed to improve the plan.

The intuition behind AIRS is as follows. When the heuristic value is small while the actual cost is large between two states, then it would be highly likely that the greedy search algorithm which generated the initial plan has failed to find a good solution, and the better plan can be quickly found using admissible search because the optimal cost would be small according to the heuristic value.

3 Window-based Plan Optimization

However, the assumptions behind AIRS may not necessarily hold true for two reasons. First, it assumes that the value $h(s_i, s_j)$ is reasonably accurate, and that $c^*(s_i, s_j)$, the optimal cost of moving from s_i to s_j, is somewhat close to to $h(s_i, s_j)$. This is not necessarily true when, for example, h fails to correctly capture the search space structure and the difference between $c(s_i, s_j)$ and c^* is very small (or even zero). In such cases, attempting to replanning the given region may not be worthwhile (because the possible gain $c-c^*$ is small) or even futile. Second, due to the same reason, c^* may be too large for the admissible replanning algorithm to find the solution within a given time limit.

In order to separate the issues which are specific to AIRS from the general framework of local search on plan segments, we first define a class of optimization technique called *Window-based Plan Optimization*.

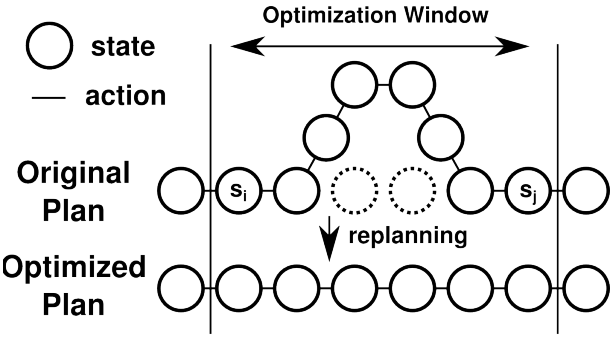

Figure 2: Simple figure describing the concept of window-based plan optimization.

In each iteration of plan refinement, Window-based optimization selects a *window* $W = (s_i, s_j)$ for the local search, which is a portion of the plan that is selected according to some criteria C, which is a free parameter of this framework. s_i, s_j are encoded as the initial state and the goal state of an input to some standard search algorithm A. If A successfully returns a better solution, it replace W with the new path. If A failed to find a better solution due to some criteria

such as the time limit, or because the resulting cost is worse than the original (note that A can be inadmissible), the solution is ignored. Window-based optimization is naturally an anytime algorithm which returns the current best solution any time it is interrupted.

A trivial baseline instance of this framework randomly selects a window of fixed size L. We call this variant **R-WIN** (Randomized Window-search). The parameter L controls the size of the window as well as the upper bound for the search conducted by the given algorithm A, so that it avoids solving the window that is beyond the capability.

The reason we adopted a randomized strategy rather than a deterministic strategy e.g. first select $W = (s_0, s_L)$, then $W = (s_1, s_{L+1})$ and so on, is to diversify the effort uniformly over the length of the plan. Otherwise, the effort put by the local search could be biased toward the beginning of the plan, while the possible refinements only exist near the end of the plan.

3.1 Empirical Evaluation of R-WIN

We evaluated R-WIN and AIRS on the International Planning Competition (IPC) benchmark instances included with the Fast Downward planner (Helmert 2006). We evaluated all methods on the IPC satisficing problem instances which are limited to STRIPS with action costs, since that is the subset of PDDL currently supported by our optimizer. All experiments in this paper were conducted on Xeon E5410@2.33GHz CPUs. We first solved these instances using Fast Downward LAMA2011 (Richter and Westphal 2010) using 15 minutes, 2GB setting. Among 958 problem instances, LAMA solved 828 problem instances in the first 15 minutes.

The resulting plans are then fed into the plan optimizers, under 2GB memory and the time limit of 15 minutes, resulting in 30 minutes of the total runtime. Since LAMA uses RWA* and produce multiple solutions, we selected the current best solution found when the time limit is reached. We evaluated the following optimizer configurations:

- LAMA (30 min) – this baseline continues to run LAMA for an additional 15 minutes (i.e., 30 minutes total) and returns the best solution found.
- Action Elimination (Nakhost and Müller 2010, AE)
- Action Dependency (Chrpa, McCluskey, and Osborne 2012, AD)
- Plan Neighborhood Graph Search (Nakhost and Müller 2010, PNGS): We used $N = 1000$ (as in the original paper).
- Anytime Iterative Refinement of a Solution (Estrem and Krebsbach 2012, AIRS)
- R-WIN: $L = 10$. Since R-WIN is a randomized algorithm, we ran the same experiments with two different random seeds in order to test the robustness of the algorithm. However, we could not observe any significant difference so we only show 1 set of results.

We implemented AE, AD, PNGS, and AIRS based on the descriptions in the original papers. However, for all search-based optimization algorithms (PNGS, AIRS, R-WIN), the replanning algorithm we used was Fast Downward with A* using the admissible LMcut heuristic (Helmert and Domshlak 2009). This differs from the original paper in that AIRS originally used Bidirectional A*(Pohl 1971), and PNGS used Dijkstra Search. This allows us to focus on the effect of the replanning strategy, as opposed to the underlying search algorithm.

Table 1 compares the ratio between the sum of the costs of the plans returned by LAMA with 15 minutes, against the results of various optimizers. Running LAMA for an additional 15 minutes results in very little improvement over the first 15 minutes, and is worse than all of the postprocessing optimizers that we evaluated. Although the difference was very small, R-WIN outperformed PNGS. This shows that a simple window-based algorithm like R-WIN can outperform a more complicated algorithms. Interestingly, AIRS was found to perform *worse* than R-WIN, a possible baseline of the Window-based optimization algorithms. Also, the polytime algorithms such as AE and AD perform reasonably good in terms of the runtime and the quality gain, even if these algorithms capture only the limited scope of optimization.

Algorithm	Harmonic Means of Ratios
LAMA(15min)	100%
LAMA(30min)	99.3%
AE	98.4%
AD	97.4%
AIRS	97.9%
PNGS	96.0%
R-WIN	**95.9%**

Table 1: Comparison of the cost-reduction ration between AE, AD, PNGS, AIRS, R-WIN, relative to bare LAMA with 15 minutes (100%). We took the harmonic means of the ratios over all problem instances. R-WIN outperformed PNGS, but with a small difference.

The complete domain-wise results in Table 2 (which also contains results for the next section) suggests that the performances of R-WIN and PNGS are significantly affected by the domain characteristics. For example, R-WIN performs better than PNGS in 12 domains out of 39 domains, although PNGS performs better than R-WIN in 18 domains. Thus, there is no overall dominance relationship between these optimization algorithms. We can also see that R-WIN fails to improve any plan on some domains such as floortile-sat11, parcprinter-08/sat11, pipesworld (both tankage and notankage versions), sokoban-sat08/11, woodworking-sat08. In these domains, we observed that R-WIN is consuming too much time on "difficult" window replanning problem instances. The same symptom was also observed in AIRS. This suggests that addressing these issues would result in a simple yet effective optimization algorithm.

3.2 CH-WIN

Since the results in the previous section suggested some possibilities for improvement within the Window-based opti-

domain	LAMA 30min.	AD	AE	AIRS	PNGS	R-WIN	CH-WIN
airport	99.9	100	100	100	100	100	**99.3**
barman11	99.9	94.8	91.0	100	100	**87.6**	96.0
blocks	100	100	87.8	91.1	80.1	84.9	**78.5**
depot	98.7	92.2	93.6	99.3	93.0	89.6	**88.7**
driverlog	99.0	97.1	96.4	98.7	93.7	92.2	**90.2**
elevators08	99.8	92.0	91.7	99.9	98.0	**86.3**	89.5
elevators11	99.9	92.7	**92.4**	100	100	95.5	97.3
floortile11	91.9	97.5	94.7	100	98.3	96.7	**80.4**
freecell	99.3	99.1	99.0	99.8	91.8	99.8	**95.1**
grid	100	100	100	100	**91.5**	92.7	91.9
logistics00	100	98.7	98.7	96.7	98.0	97.3	**94.8**
logistics98	100	99.0	98.8	99.2	99.6	98.5	**98.1**
miconic	100	100	100	89.7	89.7	90.0	**88.4**
nomystery11	100	100	100	100	100	100	**99.8**
openstacks08	**98.7**	100	100	100	100	100	100
openstacks11	**97.8**	100	100	100	100	100	100
parcprinter08	99.0	100	98.3	100	100	100	**94.2**
parcprinter11	99.0	100	100	100	100	100	**93.1**
parking11	99.8	99.5	99.5	100	100	99.3	**98.7**
pegsol08	97.1	100	100	100	100	94.9	**89.4**
pegsol11	91.7	100	100	100	100	96.0	**85.9**
pipesworld-not	97.6	99.8	93.1	98.3	93.9	100	**89.3**
pipesworld-t	97.3	100	95.3	98.6	92.8	100	**92.3**
rovers	99.9	96.8	**96.8**	99.7	99.4	98.3	98.9
satellite	99.9	99.1	98.9	97.2	98.1	**95.5**	96.6
scanalyzer08	99.3	100	100	100	98.6	96.2	**93.2**
scanalyzer11	99.3	100	100	100	98.1	96.2	**93.2**
sokoban08	95.2	100	98.4	100	93.9	99.7	**90.7**
sokoban11	94.6	100	98.2	100	93.2	100	**90.1**
tpp	99.7	99.5	99.5	98.0	98.8	95.4	**92.6**
transport08	100	94.2	**94.1**	100	99.3	95.9	97.1
transport11	99.8	92.3	**91.9**	100	100	97.5	99.6
trucks	**99.1**	100	100	100	100	100	100
visitall11	99.9	100	99.7	100	100	99.6	**99.5**
woodworking08	100	99.6	99.3	100	100	100	**88.2**
woodworking11	100	99.8	98.8	100	100	100	**90.8**
zenotravel	100	100	100	100	98.7	100	**97.3**
mean(harmonic)	99.3	98.4	97.4	97.9	*96.0*	*95.9*	**93.3**

Table 2: Comparison of the cost-reduction ratios between AE, AD, PNGS, AIRS, R-WIN and CH-WIN, relative to plain LAMA with 15 minutes (100%). The costs are the sum over all instances solved by LAMA within 15 minutes.

mization framework, we implemented a more sophisticated window selection scheme named **CH-WIN**. The name reflects the fact that it uses the ratio between the actual cost c and the heuristic value h, similar to AIRS (Estrem and Krebsbach 2012), but with several simple improvements that address the problems present in AIRS and R-WIN.

First, we improved the window selection / penalization criteria in AIRS. Although AIRS adopts a mechanism which gives a penalty to a window whose region overlaps with the previously selected windows and tries to avoid solving them, we found that the mechanism does not sufficiently deter the optimizer from duplicated work, sometimes repeatedly selecting the nearby windows. In CH-WIN, we never select the same segment twice. Windows are simply sorted into a priority queue according to the value of $h(s_i, s_j)/c(s_i, s_j)$, and no window is reinserted into the queue after being optimized. Thus CH-WIN simplifies AIRS by removing the penalty mechanism – there is no need to consider the overlaps between the currently selected window and the past windows whose replanning attempt has failed.

In fact, although it is not worthwhile to select the same segment more than twice , it is still possible to improve the segments which overlap it. There are two possible reasons that the previous attempt has failed. (1) First, the failure may be due to the large $c^*(s_i, s_j)$ which makes it too difficult for admissible algorithms to obtain a refined solution. Thus, replanning its shorter subsegments like (s_k, s_l) s.t. $i < k < l < j$ may succeed. (2) Second, the failure could be due to a successful replanning with no improvements, i.e. $c(s_i, s_j) = c^*(s_i, s_j)$. It suggests that its subsegment can be solved much quicker, which means that replanning is not so harmful. Also, it does not preclude the chance of improving a segment which overlaps it, such as (s_k, s_l) s.t. $k < i < l < j$. Thus, we consider the simplified criteria offers a wider and safer opportunity to improve the plan quality overall. As an implementation detail, each h, c, s_i and s_j is updated after the successful replanning of a window, in order to reflect the changes to the plan and the positions of states in the plan.

Second, we applied a time limit of 3 minutes for each replanning episode. This avoids the problem we observed in AIRS and R-WIN that some individual replanning attempts can consume too much time. It is possible that AIRS does not address this issue because in the original paper, AIRS was tested on problems where the replanning attempts were all relatively easy (under 0.1 second per episode in the 15 puzzle with domain-specific heuristics and grid pathfinding).

Third, along the same line as our second improvement above, we apply a maximum length to the replanning window, which is dynamically adjusted as the replanning succeeds or fails. The dynamic process has two parameters L and $\bar{L}$, and runs a binary search of the appropriate length as shown in Algorithm 1. The value of L trivially converges to some value L_∞.

We tested CH-WIN using the same experimental setting as in the previous section. The results are shown in Table 3, which shows that CH-WIN outperforms both R-WIN and PNGS algorithm. Per-domain results in Table 2 also

Algorithm 1 Binary search of L, used for adapting the size of the replanning window.

1: Initially $L = n/4$ and $\bar{L} = n$, where n is the length of the plan.
2: When CH-WIN selects the next window, it skips windows whose lengths exceed L. Skipped windows are inserted back to the end of the queue, so that future increases in L allow the optimizers to reconsider those candidates.
3: **if** replanning succeeds within the time limit, **then**
4: $\quad L \leftarrow (L + \bar{L})/2$.
5: **else**
6: $\quad L \leftarrow L/2$ and $\bar{L} \leftarrow L$.
7: **end if**

show that CH-WIN outperforms other algorithms in many domains (26 domains out of 36 domains).

Algorithm	Harmonic Means of Ratios
LAMA(15min)	100%
LAMA(30min)	99.3%
PNGS	96.0%
R-WIN	95.9%
CH-WIN	**93.3%**

Table 3: Summary of the comparison of the cost-reduction ration between PNGS, R-WIN and CH-WIN relative to bare LAMA with 15 minutes (100%). We took the harmonic means of the ratios over all problem instances. CH-WIN outperformed both PNGS and R-WIN.

4 Evaluating Sequences of Optimization Techniques

As a natural consequence of Plan Optimizer taking a plan as an input and emits a plan as an output, we can combine several different optimizer algorithms in sequence, hoping that each algorithm addresses different kind of inefficiencies / redundancy in the suboptimal plan.

This idea is not new. Aras (Nakhost and Müller 2010) applies Action Elimination to the input, then run PNGS to its output. Siddiqui and Haslum also applied BDPO2 to the results of PNGS in one of their experiments (Siddiqui and Haslum 2015). However, currently there are few in-depth analysis on the effect of combining many postprocessing optimization algorithms.

4.1 Sequences of Poly-Time Optimizers

We first evaluated two polytime algorithms (AD, AE) applied in different orders (AD+AE and AE+AD) and in iterations (AE+AD+AE, AE+AD+AE+AD, ...) to ensure that simply sequencing these polytime optimizers does not further improve the plan quality.

As shown in Table 4, repeatedly alternating between AE and AD does not improve the plan quality. Although there is a slight difference in the result depending on which optimizer is applied first, it happens only in the depot and elevators08 domains. This yields two observations: First, when we use AE and AD as a part of optimizer sequence, applying each of AE and AD once is enough to remove the inefficiency addressed by AE and AD. Second, we need to add either a search-based optimizer (such as PNGS, AIRS, CH-WIN) or a new polytime optimizer (future work) to the optimizer sequence in order to get a better plan.

	AD	AD AE	AD AE AD	AD AE AD AE	AE	AE AD	AE AD AE	AE AD AE AD
domain								
mean(harmonic)	98.4	97.4	97.4	97.4	97.4	97.4	97.4	97.4

Table 4: Result of applying poly-time optimizers iteratively to plans obtained by 15-minute runs of LAMA. The harmonic means of the ratios over all instances are shown.

4.2 Sequences Poly-time Optimizers and a Search-Based Optimizer

We next evaluated two polytime algorithms (AD, AE) applied in this order (AD+AE) with one of three search-based algorithms (AIRS, PNGS, R-WIN, CH-WIN), i.e., AD + AE + S configuration where S is one of AIRS, PNGS, R-WIN, CH-WIN.

We tested these configurations with the same resource limitations (time, memory) as in the previous experiments. Each optimizer configuration processes the input plan with a 15 minutes time limit (total, shared among all components of the optimizer shared) and 2GB memory limit. The input plans are the best results output by running LAMA for 15 minutes, with a 2GB limit on each problem instance. Polytime algorithms usually finish very quickly, and their runtimes are negligible compared to the search-based optimizers. Thus, we assume most of the 15 minutes of runtime for postprocessing is consumed by the search-based optimizers in the configuration.

As shown in the previous section, repeating AD and AE steps do not improve the quality nor result in different plans, so we do not test configurations which repeat several AD+AE sequences (such as AE+ AD+ AE+ AD+ PNGS).

Table 5 shows the results of running these configurations of multiple optimizers. Combining several optimizers behaved as expected: The results of search-based optimizers with polytime optimizers tends to be better than without them, and polytime optimizers themselves are also helped by search-based optimizers. We can see that the best performer was AE + AD + CH-WIN, a configuration which applies Action Elimination first, then applies the Action Dependency to the output of AE, then applies CH-WIN to the output of AD.

We also evaluated configurations where each of above configuration is followed by an additional applications of AE and AD (i.e. AE+ AD+S+ AE+ AD), but we could not observe any quality improvement: The harmonic-mean scores were 95.6, 94.4, 94.0, 91.8 (for S=AIRS, PNGS, R-WIN, CH-WIN, respectively), which means that the search-based optimizers we used in this paper tend not to regener-

ate the inefficiency which can be detected by the polytime optimizers. This might be a natural consequence of the underlying replanner A being an admissible (optimal) planner. Investigating the cases which use inadmissible planner as A is future work.

4.3 Sequences With Multiple Search-Based Optimizers

We finally investigated the performance of the combinations of multiple search-based optimizers (PNGS, R-WIN, CH-WIN). We focus on the synergy that can happen on several different optimizer algorithms because there are still some domains in Table 5 where PNGS and R-WIN yield better results than CH-WIN.

We do not include AIRS in this experiments because CH-WIN is an improved version of AIRS and we showed in previous sections that CH-WIN outperforms AIRS. Also, we do not need to interleave AE+AD between the runs of search-based optimizers because we showed, in the previous section, that the results emitted by the search-based optimizers usually do not contain inefficiency which can be detected by poly-time optimizers. Thus, for each search-based optimizer S_1 and S_2, we tested AE+ AD+ S_1+ S_2 (where S_1 and S_2 should be different optimizers).

In this experiment, we run AE and AD first, then S_1 for 7.5 minutes, then S_2 for 7.5 minutes. In all cases, the replanning is constrained under 2GB memory limitation, and the input was again the results of LAMA being run for 15 minutes, 2GB constraints.

The results in Table 6 show that the combination of different search-based optimizers does *not* yield better results than AE + AD + CH-WIN. This supports our claim that CH-WIN has both a proper window selection scheme (use of c/h ratio), which is lacking in R-WIN and PNGS, and a window scaling scheme (adaptive window size), which is lacking in R-WIN and AIRS, thus outperforming those algorithms by successfully focusing the replanning effort on to the region of "most room for improvements".

5 Conclusion

We proposed and evaluated a simple, window-based replanning approach to plan optimization. Although it has been conjectured that a sliding window based techniques is "unlikely to find relevant subproblems in general planning problems where the sequential plan is often an arbitrary, interleaving of separate threads" (Siddiqui and Haslum 2013), our results indicate that a window based approach is sufficient to obtain significant plan quality improvements compared to the baseline, as well as to previous methods such as PNGS. A comparison with the block-deordering strategies in BDPO2 (as well as its combinations with other algorithms) is an avenue for future work (Siddiqui and Haslum 2013).

One advantage of our approach is simplicity of implementation. CH-WIN is a pure, wrapper-based approach which uses a standard off-the-shelf planner for replanning, as well as the heuristic value computations needed for window selection. Our implementation of CH-WIN required no modi-

domain	AE AD AIRS	AE AD PNGS	AE AD R-WIN	AE AD CH-WIN
airport	100	100	100	**99.3**
barman11	91.0	91.0	85.0	**89.2**
blocks	84.2	82.9	83.9	**80.1**
depot	93.4	89.9	**87.4**	88.0
driverlog	95.6	92.3	89.6	**87.8**
elevators08	91.6	90.8	**80.5**	82.5
elevators11	92.4	92.4	**87.7**	89.8
floortile11	94.7	93.2	93.7	**78.9**
freecell	98.9	**91.2**	99.0	94.6
grid	100	**91.5**	92.7	91.9
logistics00	96.2	97.6	96.8	**93.4**
logistics98	98.3	98.5	97.4	**97.1**
miconic	89.8	89.8	89.8	**88.4**
nomystery11	100	100	100	**99.8**
openstacks08	100	100	100	100
openstacks11	100	100	100	100
parcprinter08	98.3	98.3	98.3	**94.2**
parcprinter11	100	100	100	**93.1**
parking11	99.5	99.5	98.2	**98.6**
pegsol08	100	100	94.9	**89.4**
pegsol11	100	100	92.1	**85.9**
pipesworld-not	92.6	91.6	93.1	**87.8**
pipesworld-t	93.9	90.5	95.3	**90.4**
rovers	96.7	96.8	**96.3**	96.4
satellite	96.7	97.3	**95.6**	**95.6**
scanalyzer08	100	98.6	96.4	**93.2**
scanalyzer11	100	98.5	98.2	**93.2**
sokoban08	98.4	94.4	98.1	**88.2**
sokoban11	98.2	93.7	98.2	**87.2**
tpp	97.6	98.8	93.0	**92.5**
transport08	94.1	93.1	91.1	**92.1**
transport11	91.9	91.9	**90.9**	91.8
trucks	100	100	100	100
visitall11	99.7	99.7	**99.3**	**99.3**
woodworking08	99.3	99.3	99.3	**87.6**
woodworking11	98.8	98.8	98.8	**88.9**
zenotravel	100	100	100	100
mean(harmonic)	95.6	94.4	94.0	**91.8**

Table 5: Results of using poly-time optimizers and search-based optimizers, on the plans obtained by 15 minutes runs of LAMA. The harmonic means of the ratios over all problem instances are shown. The poly-time optimizers did not harm the successive application of search-based optimizers, and the combination using our CH-WIN was the best performer, same as in the previous experiments.

	AE AD					
S_1	PNGS		R-WIN		CH-WIN	
S_2	R-WIN	CH-WIN	PNGS	CH-WIN	PNGS	R-WIN
airport	100	99.6	100	99.5	99.6	99.6
barman11	86.2	90.1	85.9	85.6	89.9	85.8
blocks	81.9	80.1	85.2	78.6	79.9	79.6
depot	88.4	88.3	87.8	88.5	88.1	87.6
driverlog	89.8	87.4	89.9	86.9	87.0	86.4
elevators08	86.5	86.3	80.0	81.2	84.1	80.3
elevators11	91.4	92.2	88.7	88.5	90.9	88.3
floortile11	90.8	78.6	86.9	78.6	78.9	78.9
freecell	91.7	91.4	98.2	98.8	91.7	95.8
grid	91.5	91.5	91.5	92.3	91.5	91.9
logistics00	96.5	94.5	96.6	94.1	94.2	94.2
logistics98	97.6	97.6	97.9	97.6	97.6	96.8
miconic	89.4	87.3	89.6	87.7	88.6	88.6
nomystery11	100	100	100	100	100	100
openstacks08	100	100	100	100	100	100
openstacks11	100	100	100	100	100	100
parcprinter08	97.5	94.5	97.8	94.2	94.2	94.2
parcprinter11	98.5	94.2	99.5	93.7	93.2	93.2
parking11	98.6	98.9	98.1	98.7	98.9	98.2
pegsol08	93.2	88.3	91.4	87.1	91.4	93.2
pegsol11	87.7	85.9	87.7	83.8	87.7	89.4
pipesworld-not	91.6	88.8	91.6	88.7	88.5	88.7
pipesworld-t	91.0	90.1	90.8	91.2	89.0	91.4
rovers	96.7	96.4	96.5	96.2	96.4	96.3
satellite	95.2	96.0	95.2	94.9	95.9	94.8
scanalyzer08	96.6	94.8	96.1	94.2	94.0	94.0
scanalyzer11	96.7	95.6	96.2	95.7	94.8	94.3
sokoban08	95.1	89.0	96.5	94.0	91.2	91.2
sokoban11	94.4	88.7	94.6	91.6	90.6	90.4
tpp	95.1	94.7	94.7	92.9	93.7	91.7
transport08	90.9	91.7	90.9	91.8	91.7	91.6
transport11	90.9	91.4	91.0	91.0	90.9	91.1
trucks	100	100	100	100	100	100
visitall11	99.4	99.3	99.4	99.0	99.4	99.5
woodworking08	98.9	97.6	98.9	89.0	89.5	89.0
woodworking11	98.8	98.8	98.8	92.4	92.5	92.4
zenotravel	100	100	100	100	100	100
mean(harmonic)	93.4	92.5	93.6	92.2	92.1	92.0

Table 6: Results of applying polytime optimizers (AE and AD) and multiple search-based optimizers (S_1 and S_2) in sequence, on the plans obtained by 15 minutes runs of LAMA. The numbers show the harmonic means of the ratios over all problem instances. Each search-based optimizer is run for 7.5 minutes under 2GB constraints.

Detailed analysis on the log file has shown that the poly-time optimizer failed to find a better plan within the time limit in many instances and simply passed the input plan to the next optimizer.

Overall, participation of CH-WIN resulted in better plans compared to those without CH-WIN, and none of these configuration outperformed the configuration AE + AD + CH-WIN (reduction ratio 91.8%), indicating that CH-WIN dominates AIRS, R-WIN, PNGS.

fications to the Fast Downward planner. The CH-WIN code is simple: aside from the simple window selection schemes described in this paper, the only thing that the CH-WIN code needs to do is to perform a forward simulation of a PDDL problem (for window start/end point construction), i.e., it is no more complicated than a simple plan validator. Thus, CH-WIN is a promising, "quick-and-dirty" approach for plan optimization.

6 Acknowledgments

This research was supported by a JSPS Grant-in-Aid for JSPS Fellows and a JSPS KAKENHI grant.

References

Balyo, T.; Barták, R.; and Surynek, P. 2012. Shortening Plans by Local Re-planning. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, 1022–1028.

Chrpa, L., and Siddiqui, F. 2015. Exploiting Block Deordering for Improving Planners Efficiency. In *Proc. IJCAI*.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012. Optimizing Plans through Analysis of Action Dependencies and Independencies. In *Proc. ICAPS*.

Estrem, S. J., and Krebsbach, K. D. 2012. AIRS: Anytime Iterative Refinement of a Solution. In *FLAIRS Conference*.

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS*.

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

Nakhost, H., and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In *Proc. ICAPS*, 121–128.

Pohl, I. 1971. Bi-directional Search. *Machine Intelligence* 6:127–140.

Ratner, D., and Pohl, I. 1986. Joint and LPA*: Combination of Approximation and Search. In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.*, 173–177.

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.

Siddiqui, F. H., and Haslum, P. 2013. Plan Quality Optimisation via Block Decomposition. In *Proc. IJCAI*.

Siddiqui, F. H., and Haslum, P. 2015. Continuing Plan Quality Optimisation. *J. Artif. Intell. Res.(JAIR)* 54:369–435.

Thayer, J. T.; Benton, J.; and Helmert, M. 2012. Better Parameter-Free Anytime Search by Minimizing Time Between Solutions. In *Proc. SoCS*, 120–128.

Xie, F.; Valenzano, R. A.; and Müller, M. 2013. Better Time Constrained Search via Randomization and Postprocessing. In *Proc. ICAPS*.