Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement

Hootan Nakhost and Martin Müller
Department of Computing Science
University of Alberta
{nakhost,mmueller}@ualberta.ca

Abstract

Compared to optimal planners, satisficing planners can solve much harder problems but may produce overly costly and long plans. Plan quality for satisficing planners has become increasingly important. The most recent planning competition IPC-2008 used the cost of the best known plan divided by the cost of the generated plan as an evaluation metric. This paper proposes and evaluates two simple but effective methods for plan improvement: *Action Elimination* improves an existing plan by repeatedly removing sets of irrelevant actions. *Plan Neighborhood Graph Search* finds a new, shorter plan by creating a plan neighborhood graph $PNG(\pi)$ of a given plan π, and then extracts a shortest path from $PNG(\pi)$. Both methods are implemented in the ARAS postprocessor and are empirically shown to improve the result of several planners, including the top four planners from IPC-2008, under competition conditions.

Improving Plan Quality

Satisficing deterministic planners can solve much harder instances than optimal planners but may generate plans that are far from optimal. Earlier planning competitions have emphasized coverage in terms of total number of problems solved, as well as raw speed. The focus of IPC-2008 was on finding the best plan with a given finite amount of resources. Much work in satisficing planning has gone into generating a high quality plan directly. Such systems output a single plan and then stop. In contrast, anytime planners such as LAMA (Richter, Helmert, and Westphal 2008; Richter and Westphal 2008) and LPG (Gerevini, Saetti, and Serina 2008) aim to quickly find a lower-quality plan, then improve it over time. The contribution of this paper are two simple but effective postprocessing methods for plan improvement: *Action Elimination (AE)* and *Plan Neighborhood Graph Search (PNGS)*. Both methods can take any valid plan as input and attempt to improve it. AE is a fast algorithm, while PNGS works in anytime fashion. Both AE and PNGS improve the performance of all the planners tested as measured by the IPC-2008 metric. In contrast to LAMA, the new methods search for local improvements "near" an existing plan. In contrast to LPG, they search in state space not plan space.

There are many ways to measure plan quality. Two popular metrics for unit cost actions are sequential plan length measured in total number of actions, and makespan, the shortest execution time of a plan if actions can be executed in parallel. The IPC-2008 metric for non-uniform action costs (including zero) was additive cost, with the total cost of a plan defined as the sum of all action costs.

Related Work

Weighted A*, or WA* (Pohl 1970), produces plans that are within a constant factor W of optimal. The LAMA planner, winner of the IPC-2008 competition, uses weighted A* with a large initial value of W to quickly produce an initial plan, then gradually reduces the weight W, while using the best found plan for additional pruning. Anytime A* (Hansen and Zhou 2007) also uses successive runs of WA*. While LAMA restarts the search from the initial state each time a solution is found, Anytime A* continues the current search with new parameters. Anytime Window A* (Aine, Chakrabarti, and Kumar 2007) uses A* within a window in the search space that moves in a depth first manner. The size of the window is increased when a solution is found.

The path improvement methods Joint, LPA* (Ratner and Pohl 1986) and ITSA* (Furcy 2006) are closely related to Plan Neighborhood Graph Search and a detailed comparison will follow later.

The LPG planner (Gerevini, Saetti, and Serina 2008) uses heuristic local search in plan space. It optimizes an objective function that measures the difficulty of resolving the inconsistencies and the estimated cost of the solution. When LPG is used as an anytime system for plan improvement, it restarts from a partial plan, obtained from the current best plan by removing some actions randomly, preferring the most expensive ones. An added numerical constraint on the cost forces the next solution to be cheaper.

For the makespan metric, the post-processing approaches of (Do and Kambhampati 2003; Veloso, Pérez, and Carbonell 1990; Bäckström 1998) aim to reduce the make-span of a given totally ordered plan by converting it to a partially ordered plan. Since these approaches do not change the set of actions in the plan, they do not improve the cost according to the other metrics above. In planning by rewriting (Ambite and Knoblock 2001), domain-specific rules rewrite a given plan into a better quality one. Rewriting rules are given by an expert or learned from training examples (Upal 1999).

Two Approaches to Plan Improvement

While there is a number of current algorithms for plan improvement in the weighted A* family, there has been no recent work on the general case when the quality of the initial plan is unknown. This is surprising since such plans are arguably most in need of improvement! The two methods AE and PNGS presented here take any plan produced by a satisficing planner and try to improve it. The methods produce no global guarantees on the solution quality. However, any known quality bound for the input plan, such as W in a plan produced by WA* with an admissible heuristic, implies a corresponding tighter bound on the improved plan.

Both AE and PNGS search for the best possible plan within a *neighborhood* of similar plans, but use different concepts of neighborhood. AE only removes actions from a given plan. PNGS exactly solves a shortest path problem in a neighborhood of a plan consisting of states close to the plan's trajectory in state space.

The rest of this paper is organized as follows: After introducing necessary notation, a greedy algorithm for Action Elimination is developed. Plan Neighborhood Graph Search is described next. The experiments evaluate these algorithms as standalone methods as well as in combination.

Notation and Background

Consider STRIPS planning with additive costs, using the following notation:

Definition 1 (Planning Task). *A planning task is a tuple* $\Pi = (\Sigma, S, s_0, A, f, G)$ *where* Σ *is a finite set of propositions,* $S \subseteq 2^{\Sigma}$ *the set of all states, and* $s_0 \in S$ *the initial state. A is the set of actions* $a(pre[a], add[a], del[a])$, *where* $pre[a]$, $add[a]$ *and* $del[a]$ *are sets of propositions containing the preconditions, positive and negative effects of a, respectively.* $f : A \rightarrow \mathbb{N}$ *is the cost function and* $G \subseteq \Sigma$ *is the goal.*

Action a *is* applicable *to state* s *if* $pre[a] \subseteq s$. *The result of applying* a *to* s *is* $\Gamma(s, a) = (s \backslash del(a)) \cup add(a)$. *The result of applying a sequence of actions* $(a_1, \ldots, a_n)$ *to a state* s *is defined recursively by* $\Gamma(s, (a_1)) = \Gamma(s, a_1)$, $\Gamma(s, (a_1, \ldots, a_n)) = \Gamma(\Gamma(s, (a_1, \ldots, a_{n-1})), a_n)$.

Definition 2 (Plan). *Let* $\Pi = (\Sigma, S, s_0, A, f, G)$ *be a planning task, and* $\pi = (a_1, \ldots, a_n)$ *an action sequence.* π *is a plan for* Π *iff* $G \subseteq \Gamma(s_0, (a_1, \ldots, a_n))$. *The cost of* π *is the sum of action costs,* $cost(\pi) = \Sigma_{i=1}^{n} f(a_i)$.

Action Elimination

Given a plan π, the goal of Action Elimination is to find a shorter plan by removing actions from π.

Definition 3 (Reduction). *Let* Π *be a planning task,* π *a plan for* Π, *and* π' *a subsequence of* π. π' *is a* reduction *of* π, *denoted by* $reduct(\pi, \pi')$, *iff* π' *is also a plan for* Π.

Definition 4 (Minimal Reduction). *Let* π *be a plan and* π' *be a reduction of* π. π' *is a* minimal reduction *of* π *if for every* π'' *such that* $reduct(\pi, \pi'')$, $cost(\pi') \leq cost(\pi'')$.

A minimal reduction is a lowest-cost plan that can be achieved by removing actions. Finding a minimal reduction can be difficult: the corresponding decision problem is NP-hard (Nakhost and Müller 2010).

Algorithm 1 Action Elimination

```
Input Initial State $s_0$, plan $\pi = (a_1, \ldots, a_n)$, and
        goal condition $G$
Output A plan reduction
  $s \leftarrow s_0$
  $i \leftarrow 1$
  repeat
    mark $a_i$ {try to remove $a_i$}
    $s' \leftarrow s$
    for $j \leftarrow i + 1$ to $length(\pi)$ do
      if $a_j$ is not applicable to $s'$ then
        mark $a_j$
      else
        $s' \leftarrow \Gamma(s', a_j)$
      end if
    end for
    if $s'$ satisfies $G$ then
      remove marked actions from $\pi$ {commit}
    else
      unmark all actions
      $s \leftarrow \Gamma(s, a_i)$
    end if
    $i \leftarrow i + 1$
  until $i > length(\pi)$
  return $\pi$
```

A greedy Algorithm for Action Elimination

Action Elimination iteratively improves a given plan $\pi = (a_1, \ldots, a_n)$ by computing a plan reduction in each iteration. The details are given in Algorithm 1. Starting from a_1, the algorithm tentatively tries to remove one action a. After removing a, all other actions that lose their support - at least one of their preconditions becomes unsatisfied - are removed from the plan. If the reduced sequence remains a solution, the algorithm commits to this new plan. Otherwise, the plan is restored to the state before a was removed. The process continues until all actions in the remaining plan have been tried. Validating a single reduction takes $O(n \times p)$ time, where p is the maximum number of preconditions of an action. The time complexity of the whole algorithm is $O(n^2 \times p)$.

Algorithm 1 is just one specific, simple implementation of the idea of using successive plan reductions and can not identify all the removable actions (Nakhost and Müller 2010). In general, different reduction sequences do not necessarily lead to a unique irreducible plan. For example, if the original plan contains two redundant but different ways of achieving the same goal, a sequence of reductions could remove either one (but not both).

Plan Neighborhood Graph Search

Most state of the art planners behave in a "greedy" way in terms of a heuristic function. They only examine a tiny subset of the state space, following narrow paths guided by their heuristic. In contrast, the search of optimal planners is much broader since A* with admissible heuristics needs to expand every state with small enough f-value. Plan Neighborhood Graph Search (PNGS) takes a middle ground between these two approaches. The plan neighborhood graph represents a subset of the state space "near" the existing plan that is wider than the path searched by greedy planners. Like optimal planners, it finds the best possible solution in a search space. However, like satisficing planners, this search is limited to a small part of the whole state space. PNGS uses local search around the plan trajectory to build the neighborhood graph, then extracts a shortest path from this graph.

Let M be a deterministic graph search method, such as breadth-first or best-first search. M must be able to expand the graph of a finite search space one node at a time from a given start state s_0 to generate a sequence of states $(s_0, s_1, s_2, \dots, s_n)$. To be useable in PNGS, M must provide a method $edgeto(s)$ that returns the edge along which state s was most recently reached in the search.

For a given exploration limit L, let $L' \leq L$ be the number of states actually expanded in the search from some start point s_0. Let $v(s_0, M, L) = \{s_i | 0 \leq i \leq L'\}$ be the set of all these states and $e(s_0, M, L) = \bigcup_{i=1 \dots L'} edgeto(s_i)$ be the set of directed edges generated in this search.

Neighborhood graph search expands a given *seed graph* $SG = (V, E)$ by running M from each start state in V with exploration limit L. The *neighborhood graph* of SG is defined as $NG(SG, M, L) = (\bigcup_{x \in V} v(x, M, L), E \cup \bigcup_{x \in V} e(x, M, L))$. Algorithm 2 gives pseudocode.

Algorithm 2 Computation of Neighborhood Graph

```
Input A subgraph $(V, E)$ of a state space with
      $V = \{v_0, \dots, v_n\}$, $E \subseteq V \times V$, nonnegative integer
      $L$, and search method $M$
Output The graph $NG(V, M, L)$

  $V' \leftarrow V$
  for $i \leftarrow 0$ to $n$ do
    $M.initialize(v_i)$ {search neigborhood of $v_i$}
    for $j \leftarrow 1$ to $L$ do
      $s \leftarrow M.get\_next\_state()$
      if $is\_null\_state(s)$ then
        return $(V', E)$
      end if
      $V' \leftarrow V' \cup s$
      $E \leftarrow E \cup M.edgeto(s)$
    end for
  end for
  return $(V', E)$
```

Let $\pi = (a_1, \dots, a_n)$ be a plan, $S_\pi = \{s_0, \dots, s_n\}$ the set of all states visited when executing π, with $s_i = \Gamma(s_0, (a_1, ..., a_i))$ for $0 < i \leq n$, and $E_\pi = \{(s_0, s_1), \dots, (s_{n-1}, s_n)\}$ the edges linking successive states in the plan. With M and L defined as above, the *L-plan neighborhood graph* of π is defined as $PNG(\pi, M, L) = NG((S_\pi, E_\pi), M, L)$. Informally, $PNG(\pi, M, L)$ contains the original seed plan augmented by the union of the neighborhoods constructed using M around each state along the plan π.

The number of vertices in $PNG(\pi, M, L)$ is bounded by $(L + 1) \times (n + 1)$. While building a neighborhood graph, all goal states are identified. A lowest-cost path from s_0 to a goal state in the graph is built by a standard Dijkstra-type shortest path algorithm. If the search method M uses forward search, backward chaining from the goal states works well here since the branching factor in regression is often much smaller. For backwards plan extraction, the priority queue in Dijkstra's algorithm is initialized with all goal states in $PNG(\pi, M, L)$.

A simple anytime version of PNGS can be implemented by iteratively doubling the exploration limit L up to a resource limit. Each iteration starts with the best plan from the previous iteration as seed plan. One benefit of the exploration limit L is that it corresponds directly to the amount of resources used by the search method M. Methods such as breadth-first and best-first search need time and memory at most linear in the number of states.

The notion of plan neighborhood graph can be extended to multiple input plans as well as multiple local search methods. For multiple input plans, compute the neighborhood graph of the union of all input plans. If S_{π_0}, E_{π_0}, S_{π_1} and E_{π_1} are the states and action edges of plans π_0 and π_1, then $PNG(\pi_0 \cup \pi_1, M, L) = NG((S_{\pi_0} \cup S_{\pi_1}, E_{\pi_0} \cup E_{\pi_1}), M, L)$. Different search methods M_0 and M_1 can be used to construct a merged neighborhood combining the expansion strategies of each method as $PNG(\pi, \{M_0, M_1\}, L) = PNG(\pi, M_0, L) \cup PNG(\pi, M_1, L)$.

Extended neighborhood graphs utilize several input plans and/or search methods in order to find a better plan. Using multiple input plans allows PNGS to search near good-quality fragments of several different plans. Multiple search methods may allow better exploration of the state space.

Local Search Methods for PNGS

The experiments reported here use either a single search method, M_{A*}, or a combination of two search methods $M_{A*} + M_{bbfs}$: M_{A*} is derived from the baseline uniform cost search algorithm from the optimal track of IPC-2008 (Helmert, Do, and Refanidis 2008). It performs a "blind" A* search with the heuristic h set to 0 for goal states and to the minimum action cost in the problem for other states. However, as in LAMA (Richter and Westphal 2008), M_{A*} is modified to better deal with the zero cost actions present in several competition domains. Since blind A* never expands any other action as long as zero cost actions are available, all action costs are increased by 1 while building the neighborhood graph. For extracting the shortest path, they are reset to the true action cost to guarantee that the returned plan's cost never exceeds the input plan's cost.

The combined search $M_{A*} + M_{bbfs}$ uses forward M_{A*} search as well as backward breadth first search (bbfs). Bbfs generates predecessor states and actions that lead to a given state, ignoring action costs.

Comparison of PNGS with Related Work

Joint and LPA* (Ratner and Pohl 1986) improve a given plan by using an optimal solver. The optimal solver searches for shortcuts between any pair of states that are a fixed distance d apart in the input plan. In contrast to these approaches that redefine the goal state for each search, PNGS always uses the original goal states of the planning problem for its search. Another key difference is that instead of searching for each shortcut in isolation, PNGS builds the complete neighborhood graph before extracting a shortest path. In the example in Figure 1, building a neighborhood graph improves on separate searches. Here, M is the A* algorithm with the blind heuristic, $L = 4$, and the input plan has three states. When A* is run from each point separately, it fails to improve the input plan, as in Figures 1.b and 1.c. However, PNGS improves the cost by 5 units.

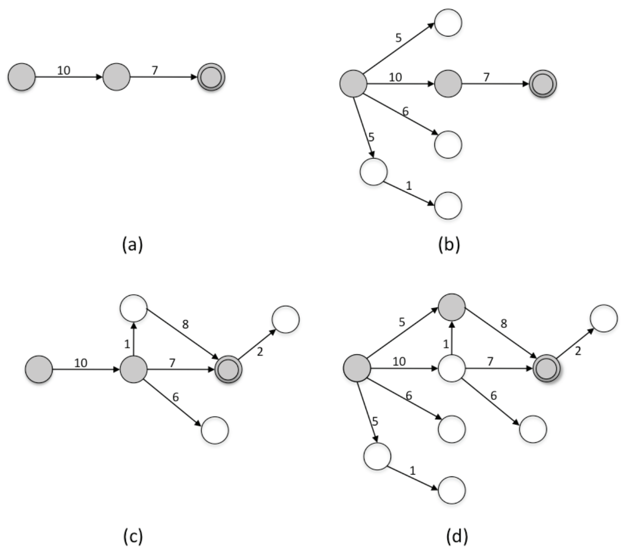

Figure 1: (a) The input plan. (b) and (c) Separate local searches fail. (d) The neighborhood graph contains an improved plan.

ITSA* (Furcy 2006) improves a given path in a graph using an A* search restricted to a tunnel near the given path π. The tunnel contains all states s with $dist(s, \pi) \le d$, where $dist(s, \pi)$ is defined as the minimum cost path from any state in π to s. ITSA* successively increases d in each iteration and terminates when a memory limit is exceeded. In (Furcy 2006), ITSA* was tested on problems with unit-cost actions, setting $d = 0, 1, 2, \cdots$. For the experiments in this paper on domains with non-uniform costs, d was set to the minimum distance among all unexplored states, as is standard practice in iterative deepening A* with nonunit costs. Each iteration runs until the first goal state is expanded.

Compared to ITSA*, PNGS uses a different search control and separates neighborhood creation from search. In contrast to the L parameter in PNGS, the search effort of ITSA* iterations can not be easily predicted from the d parameter in domains with nonuniform branching factor. ITSA*'s distance function can also lead to an unbalanced expansion at different points along the input plan, since its number of states expanded corresponds to the number of low-cost paths available. ITSA* expands many more nodes in regions where many cheap actions are available.

Building and searching the neighborhood simultaneously as in ITSA* allows some more pruning. One advantage of the two phase computation in PNGS is that different action costs can be used in each phase, which works better for domains with zero-cost actions. The option to merge neighborhoods generated by different search methods with complementary strengths is also useful.

Different Requirements for Shortest Path vs Path Improvement Algorithms

Increasing action costs by 1 for building the neighborhood graph in M_{A*} helps address the problem of zero action costs. However, if there is a big gap between the costs of cheap and expensive actions, blind A* still heavily favors expanding cheap actions. This bias is good for finding a shortest path to a goal, but problematic for the current task of improving a given path. Alternatives to both cheap and expensive actions need to be explored in order to find improved plans. This issue was discovered late in the course of this research and has not yet been addressed satisfactorily.

Experiments

The ARAS plan postprocessor implements Action Elimination and PNGS on the basis of the Fast Downward (FD) (Helmert 2006) framework. M_{A*} and M_{bbfs} are implemented as local search methods. For direct comparison, ITSA* was implemented in the same environment. Increased action costs are also used in ITSA* to avoid problems with zero cost actions. Therefore, it is not guaranteed that ITSA* returns a plan of equal or less cost. ARAS supports propositional PDDL2.2, excluding derived predicates, as well as action costs in PDDL3.1. ARAS and LPG were used to improve the results of the ARVAND (Nakhost and Müller 2009) and FF (Hoffmann and Nebel 2001) planners in the IPC-2004 domains Pipesworld Tankage, Pipesworld NoTankage, Airport and Satellite. Further, ARAS is compared to ITSA* in all IPC-2008 domains on plans produced by ARVAND, FF, and the top four planners from the competition: LAMA, FF$_{sa}$, FF$_{ha}$, and C3. Currently, LPG does not support IPC-2008 domains. The input plans for ARAS were generated by a single run of the latest available version of each planner. Tests used a 2.7 GHz AMD processor with 4GB memory and 30 minutes time limit per problem.

Experiment 1: Postprocessing for IPC-2008 Domains

Tests used the IPC-2008 scoring function, with the cost of the best plan produced by any satisficing planner at the IPC-2008 competition divided by the cost of the generated plan. Unlike the competition itself, and in order to measure progress since then, a plan that is better than the best IPC-2008 plan achieves a score higher than one.

For the planners returning a single plan, FF, FF$_{sa}$, FF$_{ha}$ and C3, the planner is run until it finds a solution. The remaining time up to 30 minutes total is used to improve the plan with ARAS or ITSA*.

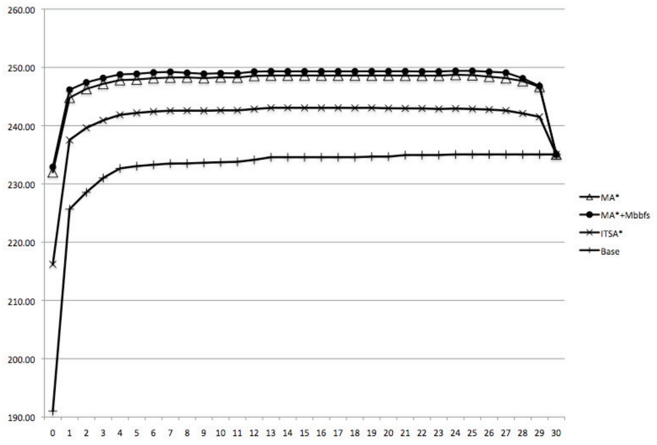

Figure 2: Total IPC-2008 score for varying cutoff times combining LAMA with M_{A*}, $M_{A*} + M_{bbfs}$ and ITSA*

Both LAMA and ARVAND can run in an anytime setting. Given both an anytime planner and an anytime postprocessor, a preliminary experiment was run to determine a reasonable allocation of time between them as follows: first, the planner is run until a fixed cutoff time is reached. If no solution is found yet, it is kept running until the first solution is found. Next, the postprocessor is used to improve the planner's best generated plan until the 30 minute timeout. The cutoff time is varied from 0 to 30 minutes in 1 minute intervals. Figures 2 and 3 show the total scores of LAMA and ARVAND over all IPC-2008 domains, when combined with the postprocessors M_{A*}, $M_{A*} + M_{bbfs}$, and ITSA*. For comparison, the baseline shows the anytime planner stopped at the cutoff time without any postprocessing. For both LAMA and ARVAND, the PNGS methods outperform ITSA*. $M_{A*} + M_{bbfs}$ and M_{A*} are very close for ARVAND. $M_{A*} + M_{bbfs}$ is slightly superior for LAMA. The best schedule for LAMA is 24 minutes (or until the first plan is found) for the planner followed by 6 minutes for ARAS, while for ARVAND the optimum is at 18 + 12 minutes. For both planners, the performance curve is almost flat for cutoff times ranging from about 7 to 26 minutes.

The results for all tested planners on IPC-2008 are summarized in Figure 7. For each planner/postprocessor pair the total score, and the score obtained in each domain is shown. LAMA and ARVAND use the cutoff times determined above. Cybersecurity is included in the totals, but no detail graph is shown; in this domain, postprocessors did not improve any plan except some generated by ARVAND.

The total scores shown in the bottom right of Figure 7 illustrate that postprocessing would have provided an advan-

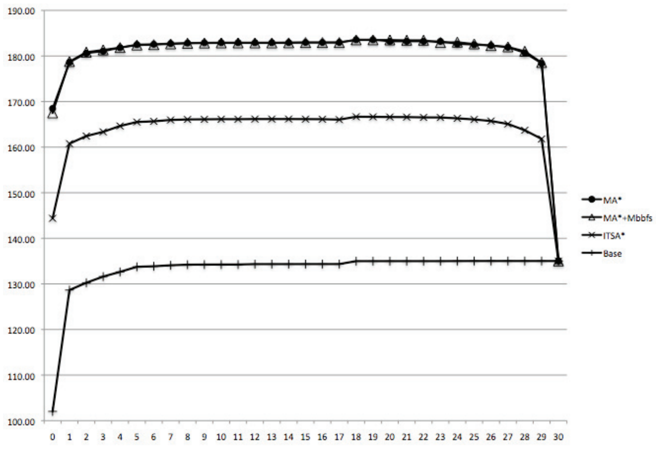

Figure 3: ARVAND with M_{A*}, $M_{A*} + M_{bbfs}$ and ITSA*

tage in the IPC-2008 competition: any of the planners that took places 3-5, FF$_{sa}$, FF$_{ha}$, and C3, would have improved to second place. Both ARAS and ITSA* find substantial improvements for many LAMA and FF plans as well, advancing the state of the art.

ARAS seems to be most effective on problems consisting of several loosely coupled subtasks. In these domains, due to low interaction between different parts of a plan, effective local improvements are possible. For example, all postprocessors perform very well in the transportation domains Transport and Elevator. In other domains, results vary greatly by planner. Postprocessing in Pegsol gains more than 10 points for FF variants and C3, and 4 points for ARAS. However, there is very limited scope for improvement for LAMA, since it already solves 27 out of 30 tasks optimally in this domain.

In PNGS, $M_{A*} + M_{bbfs}$ outperforms M_{A*}, especially in domains where local improvements are effective. Although the size of the largest neighborhood graph is equal for both search methods in these experiments, their structure is totally different. In $M_{A*} + M_{bbfs}$, the expanded states are closer to the plan, which contributes to finding better shortcuts. In contrast, in Openstacks such local improvements are hard. Actions that affect the total cost - adding a stack - completely change the search neighborhood. Most improvements are found for smaller tasks where the largest neighborhood graph contains a new goal state. Here, using all memory for M_{A*} works better than dividing it between M_{A*} and M_{bbfs}.

Both M_{A*} and $M_{A*} + M_{bbfs}$ usually outperform ITSA*, which has trouble when there are large cost differences between actions. For example, in Transport, *pick up* and *drop* have unit cost, while the distance-dependent cost of *drive* is usually much larger. ITSA* tends to explore sequences of many cheap actions, but largely ignores crucial *drive* actions. For example in Transport-14 the cheapest driving action has cost 12, and ITSA* reached a maximum $d = 71$, while the neighborhood graph of PNGS with $M_{A}*$ contained some nodes up to a cost of 253 from the input

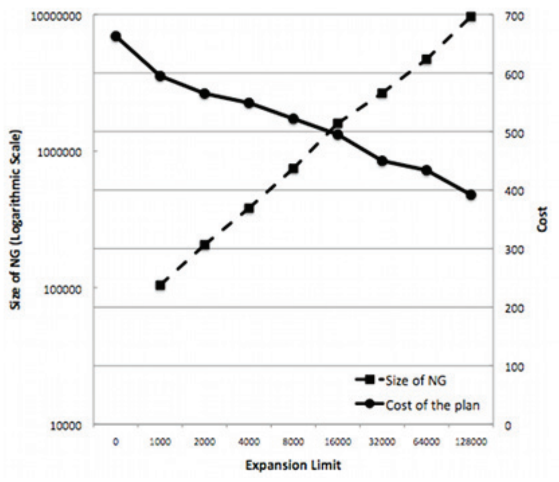

Figure 4: Plan cost and size of neighborhood graph for M_{A*} when varying the expansion limit in Elevators-22.

graph. It found a solution of overall cost 2217 compared to ITSA*'s 2617.

Figures 4 and 5 show the effect of varying expansion and distance limit using Elevators-22 as an example. The input plan was generated by LAMA and has a cost of 663. The size of the neighborhood graph in PNGS grows linearly with the expansion limit. The growth rate of ITSA* varies depending on the average branching factor in the explored regions at each iteration.

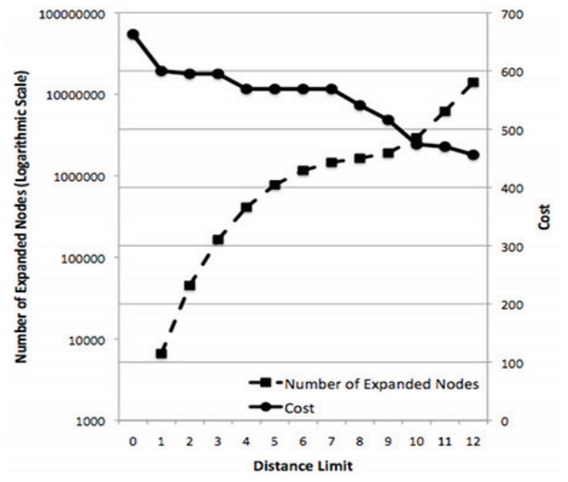

Figure 5: Plan cost and nodes expanded by ITSA* with varying d in Elevators-22.

Action Elimination

Figure 6 reports results for two configurations of ARAS that use Action Elimination: AE represents a single run of Action Elimination. PNGS + AE* runs PNGS and Action Elimination alternately: AE is used before each iteration of PNGS. $M_{A*} + M_{bbfs}$ are used as search methods in PNGS. This combination works better than either AE or PNGS alone. Running AE alternately helps to remove actions that are no longer necessary due to reductions made by the previous iteration of PNGS. For example, if PNGS replaces a sequence of actions s with a less expensive alternative, then previous actions that were supporting propositions used by s may become redundant. PNGS cannot easily identify such redundancies since paths excluding these actions do not often hit another state in the plan; usually all nearby states in the plan already contain the effects generated by earlier, now redundant actions.

On average, a PNGS run consists of 10 to 12 iterations and each iteration takes 30 seconds. AE is much faster: the average time for a single run is less than a second.

Out of the total of 270 instances tested, ARAS with PNGS + AE* improved the best previously published results for 60 instances. See (Nakhost and Müller 2010) for a detailed listing and discussion.

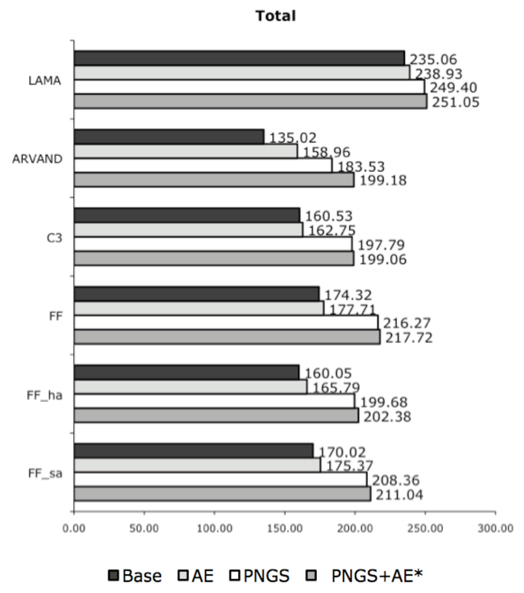

Figure 6: IPC-2008 scores of planners LAMA, ARVAND, FF, FF$_{sa}$, FF$_{ha}$, and C3 with ARAS versions AE, PNGS, PNGS + AE*.

Experiment 2: IPC-2004 - ARAS vs LPG

Table 1 summarizes the results for IPC-2004 with an IPC-2008-like metric: cost of the best plan computed in all experiments divided by cost of the generated plan. ARVAND plans were generated by a single run of the planner. LPG results are averaged over five runs. The timeout for planning and then postprocessing was set to 30 minutes total.

ARAS performs much better than LPG in improving the longer plans generated by ARVAND. The results are close for FF-generated plans, with a slight overall edge for ARAS. LPG and ARAS have different strengths since they search different spaces. Long plans with a large branching factor in plan space affect LPG much more than ARAS, while

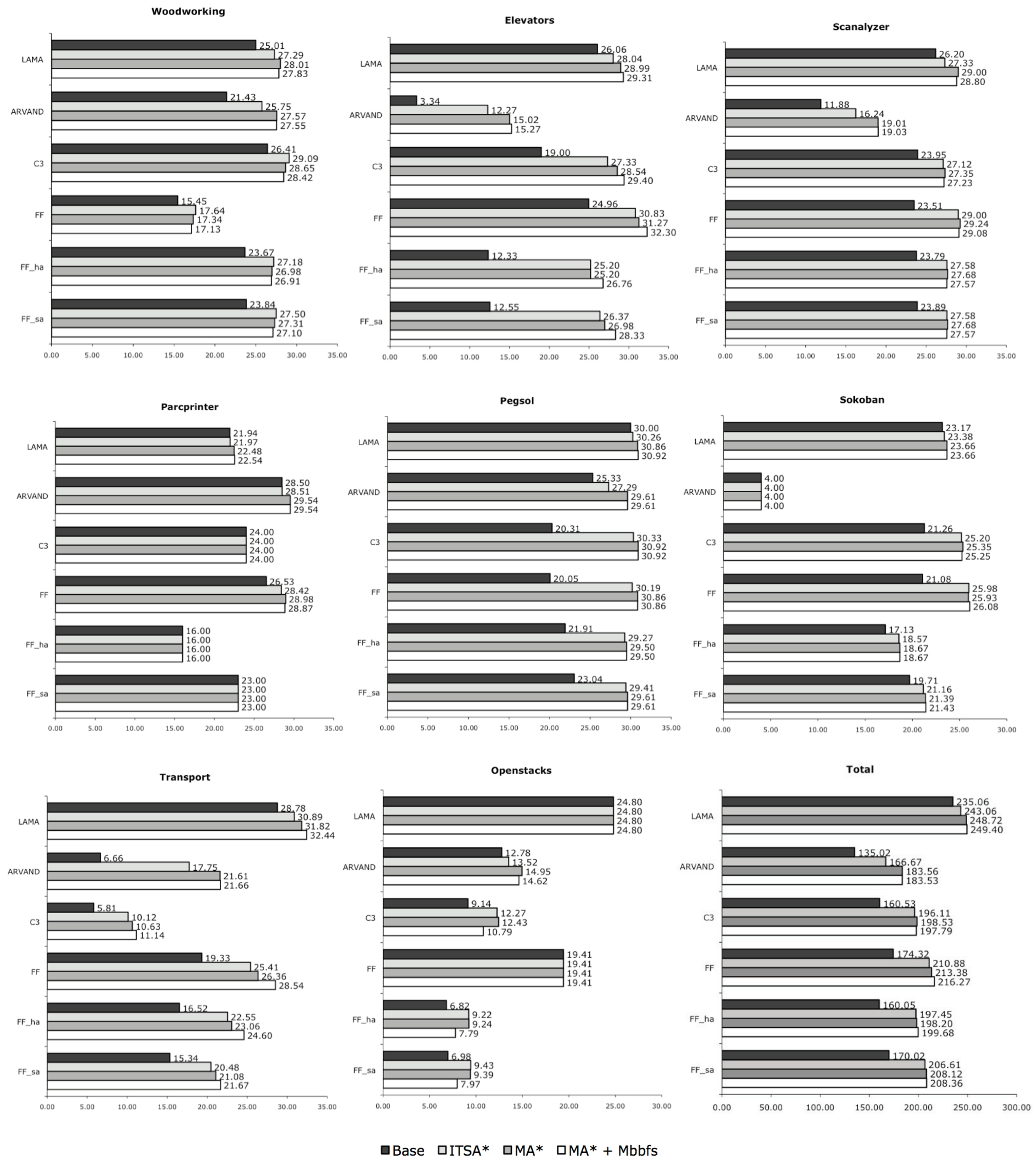

Figure 7: IPC-2008 scores of planners (LAMA, ARVAND, FF, FF$_{sa}$, FF$_{ha}$, and C3) combined with no postprocessors (base), ITSA*, ARAS M_{A*} and ARAS $M_{A*} + M_{bbfs}$. Total scores include Cybersecurity.

ARVAND

Postprocessor	No Tankage	Tankage	Airport	Satellite	Total
None	15.26	16.20	44.6	5.01	81.07
LPG	41.43	27.22	**46.53**	32.43	147.62
AE	27.73	21.81	44.6	16.11	110.25
PNGS	44.26	39.98	45	23.68	152.92
PNGS+AE*	**45.6**	**42.47**	45	**33.62**	**166.69**

FF

Postprocessor	No Tankage	Tankage	Airport	Satellite	Total
None	25.57	16.88	35.2	33.9	111.55
LPG	33.01	18.37	**36.47**	**35.45**	123.30
AE	26.39	16.93	35.2	34.58	113.1
PNGS	34.66	21.45	35.6	34.67	126.38
PNGS+AE*	**34.81**	**21.45**	35.59	34.98	**126.83**

Table 1: Combining ARVAND and FF with ARAS (AE, PNGS, PNGS+AE*) and LPG in four IPC-2004 domains.

a large branching factor in state space does not necessarily slow down LPG's search in plan space. Apart from the search space, the heuristic search in LPG is better suited to find global alternatives for good quality plans generated by planners such as FF, than to finding local improvements in a long ARVAND plan.

The results for ARVAND in Satellite are interesting. In this domain, ARVAND generates solutions with many unnecessary actions. LPG, focusing more on causal relations, is much better than PNGS in removing irrelevant actions. However, the combination of action elimination and PNGS can beat LPG: action elimination identifies irrelevant actions while PNGS searches for shortcuts.

Conclusions and Future Work

Experiments with the two plan improvement methods implemented in ARAS, Action Elimination and Plan Neighborhood Graph Search, show substantial improvements of a large variety of plans and for all tested planners. The main limitation of both methods is that they can only find local improvements near the previous plan. This approach is ineffective in domains such as Cybersecurity or Openstacks. There are many promising directions for future work:

- Find more reductions in Action Elimination.
- Adapt the search effort per node in PNGS.
- Try PNGS with multiple input plans.
- Investigate the effect of macros on plan improvability.
- Focus more on avoiding expensive actions in PNGS.

Acknowledgments

The authors wish to thank Malte Helmert for providing the source code for FD, and the anonymous referees for their valuable advice. This research is supported by a Provost Doctoral Entrance Award funded by the University of Alberta, and by grants from iCORE, the province of Alberta's Informatics Circle of Research Excellence, and NSERC, the Natural Sciences and Engineering Research Council of Canada.

References

Aine, S.; Chakrabarti, P. P.; and Kumar, R. 2007. AWA* - a window constrained anytime heuristic search algorithm. In *IJCAI*, 2250–2255.

Ambite, J. L., and Knoblock, C. A. 2001. Planning by rewriting. *JAIR* 15:207–261.

Bäckström, C. 1998. Computational aspects of reordering plans. *JAIR* 9:99–137.

Do, M. B., and Kambhampati, S. 2003. Improving temporal flexibility of position constrained metric temporal plans. In *ICAPS*, 42–51.

Furcy, D. 2006. ITSA*: Iterative tunneling search with A*. In *AAAI Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*, 21–26.

Gerevini, A.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *JAIR* 28:267–297.

Helmert, M.; Do, M.; and Refanidis, I. 2008. International Planning Competition-2008, Deterministic Part. Available at `http://ipc.informatik.uni-freiburg.de/`.

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.

Nakhost, H., and Müller, M. 2009. Monte-Carlo exploration for deterministic planning. In *IJCAI*, 1766–1771.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement - extended version. Technical Report TR 10-01, Dept. of Computing Science. University of Alberta.

Pohl, I. 1970. Heuristic search viewed as pathfinding in a graph. *Artificial Intelligence* 1(3):193–204.

Ratner, D., and Pohl, I. 1986. Joint and LPA*: combination of approximation and search. In *AAAI*, 173–177.

Richter, S., and Westphal, M. 2008. The LAMA planner. Using landmark counting in heuristic search. `http://ipc.informatik.uni-freiburg.de/Planners`.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.

Upal, M. A. 1999. Learning rewrite rules to improve plan quality. In *AAAI*, 984.

Veloso, M. M.; Pérez, M. A.; and Carbonell, J. G. 1990. Nonlinear planning with parallel resource allocation. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. Morgan Kaufmann.