Spock: Fast Downward Stone Soup with Redundant Action Elimination

Mauricio Salerno[1], **Raquel Fuentetaja**[1], **Jendrik Seipp**[2]

[1]Universidad Carlos III de Madrid, Leganés, Madrid, Spain
[2]Linköping University, Linköping, Sweden
msalerno@pa.uc3m.es, rfuentet@inf.uc3m.es, jendrik.seipp@liu.se

Abstract

In this planner abstract, we introduce Spock, a submission to the sequential satisficing track of IPC 2023. Spock runs the Fast Downward Stone Soup 2018 portfolio and reduces each discovered plan by eliminating redundant actions. The action elimination process is founded on a reformulation of the given planning task. Solving this novel planning task with an optimal planner identifies the most costly set of redundant actions within a plan, and removing it yields a cheaper plan and a lower upper bound for the next portfolio component. In numerous instances, the action elimination procedure can be executed in mere tenths of a second. As a result, Spock offers an affordable post-planning optimization step that enhances the performance of satisficing planners without significantly increasing computational demands.

Fast Downward Stone Soup

Fast Downward Stone Soup (FDSS) (Helmert, Röger, and Karpas 2011) is a portfolio planner based on the Fast Downward planning system (Helmert 2006). We use the version that competed in the sequential satisficing track of the IPC 2018, extending it with an action elimination module that removes redundant actions in the found plans. Since we do not make any changes to the portfolio, we only describe it briefly here, and refer to the original Stone Soup paper (Helmert, Röger, and Karpas 2011) and the 2018 planner abstract for further information (Seipp and Röger 2018).

Fast Downward Stone Soup receives the following inputs to build a portfolio: a set of planning algorithms, a set of training instances, and evaluations for each pair of algorithm and training instance (i.e., running time and plan cost). With this information, the Stone Soup algorithm chooses a subset of the input planning algorithms, and assigns each of them a time limit. The chosen algorithms are executed sequentially, and the cost of the best plan found at any step is used to perform pruning based on *g values* for subsequent planning algorithms.

What is a Spock?

In the Star Trek franchise, the character of Spock is known for his logical and analytical approach to problem-solving. He often identifies and eliminates redundant or unnecessary steps in complex procedures in order to streamline processes and make them more efficient. In this work, in a Spockian fashion, we take the Fast Downward Stone Soup planner and eliminate any existing redundancy in each new plan found.

For tasks without zero-cost actions, an optimal plan will not contain any redundant actions, but numerous works have shown that modern satisficing planning systems do generate plans with redundant actions (Chrpa, McCluskey, and Osborne 2012b,a; Balyo, Chrpa, and Kilani 2014; Med and Chrpa 2022). The process to identify and remove redundant actions is typically fast, compared to the time needed to find a plan. Doing this provides not only plans without redundancy, which intuitively is always desired, but also reduces the plan cost by removing unnecessary steps.

Redundant Actions and Plan Justification

Intuitively, redundant actions in plans are those actions that can be removed from a plan without affecting its validity. This notion is known as *plan justification* (Fink and Yang 1992). Fink and Yang define multiple types of plan/action justifications, but the strongest of the three is called *perfect justification*. Put simply, perfectly-justified plans are those for which no subsequence of actions can be removed from the plan without invalidating it. More formally, we say that a plan without any *plan reductions* is perfectly justified.

Definition 1 (Plan Reduction) *Let π be a plan for a planning task Π and ρ be a subsequence of π with $|\rho| < |\pi|$. ρ is a plan reduction of π if and only if ρ is also a plan for Π.*

Definition 2 (Perfectly-Justified Plan) *A plan π for the planning task Π is perfectly justified if and only if there is no plan reduction of π.*

Thus, if there is at least one plan reduction of a plan, the plan is not perfectly justified. The problem of finding the cheapest perfectly justfied plan reduction of a given plan π of a planning task Π is called Minimal Reduction (MR) (Nakhost and Müller 2010; Balyo, Chrpa, and Kilani 2014).

The main goal of action elimination is to find useful plans without redundant actions, but this process clearly reduces the plan cost as a side effect. We exploit this potential cost reduction and use a solver for the minimal reduction process as a post planning optimization step for each plan found by the portfolio. This process not only (potentially) finds a cheaper plan without redundancy, but provides a better (lower) bound with which pruning based on g values can be performed on subsequent planner runs.

Minimal Reduction as Planning

We now introduce a compilation that encodes the MR problem as a planning task. Given a planning task and a plan, we define a new planning task that can eliminate subsequences of redundant actions from the plan. We encode the new task in a way that allows to either keep or skip each plan action. In the following, we consider the same action occurring at different positions in an action sequence as different actions. For this, we slightly abuse notation and let $a_i \in \pi$ represent that action a_i is at position i in π.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan for Π. Let $F_\pi = \bigcup_{a_i \in \pi} \mathit{pre}(a_i) \cup \mathcal{G}$ be the set of facts that appear either in the precondition of an action in π or in $\mathcal{G}$. Now, we create a new planning task $\Pi^{skip} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$, where:

- $\mathcal{V}' = \{v \in \mathcal{V} \mid |\mathcal{D}'(v)| > 1\} \cup \{pos\}$, where we keep variables v from the original task but with a redefined domain $\mathcal{D}'(v)$, only when $\mathcal{D}'(v)$ contains more than one value; and there is an additional variable pos with $\mathcal{D}(pos) = \{0, \ldots, n\}$ to track the last action from the original plan considered at any given state. For a variable v, $\mathcal{D}'(v)$ contains only the values of v that appear in F_π and an additional value θ, representing an *irrelevant value*. The value of a variable is irrelevant when it is set by either the initial state or the effect of some action in the plan, but the corresponding fact is not in F_π. Thus, $\mathcal{D}'(v) = \{d \in \mathcal{D}(v) \mid \langle v, d \rangle \in F_\pi\} \cup \Theta$, where $\Theta = \{\theta\}$ if $\langle v, d \rangle \notin F_\pi$ but $\langle v, d \rangle \in \mathcal{I}$ or there exists $a_i \in \pi$ with $\langle v, d \rangle \in \mathit{eff}(a_i)$, and $\Theta = \emptyset$ otherwise. The irrelevant value is used when the value of a variable is changed to a value which is not relevant to the plan. Just removing such effects is not enough because the variable does not maintain its previous value after the application of the action. This potentially reduces the size of $\mathcal{D}'(v)$ compared to $\mathcal{D}(v)$ since all facts in the effects of any action in π but not in F_π are represented by the single fact $\langle v, \theta \rangle$.
- $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid 1 \leq i \leq n\}$, where there is a new action a'_i for every action a_i in the plan and a skip action for every plan position. Actions a'_i are defined as: $pre(a'_i) = pre(a_i) \cup \{\langle pos, i-1 \rangle\}$ and $\mathit{eff}(a'_i) = \{\tau(f) \mid f \in \mathit{eff}(a_i)\} \cup \{\langle pos, i \rangle\}$, where the effects of the new action are the same as those of the original action, but changing the variable value to the irrelevant one for those facts not used in action preconditions or goals. Formally, $\tau(\langle v, d \rangle)$ is $\langle v, d \rangle$ if $\langle v, d \rangle \in F_\pi$ and $\langle v, \theta \rangle$ otherwise. The $skip_i$ actions just increase the value of pos from $i-1$ to i. They have zero cost, while the a'_i actions maintain the cost $c(a_i)$.
- $\mathcal{I}' = (\mathcal{I} \cap F_\pi) \cup \{\langle v, \theta \rangle \mid v \in \mathcal{V}', \langle v, d \rangle \in (\mathcal{I} \setminus F_\pi)\} \cup \{\langle pos, 0 \rangle\}$ contains the facts from the original initial state that are relevant for the plan, and relevant variables with an irrelevant initial value are set to the *irrelevant value*.The *pos* variable is initialized to zero.
- $\mathcal{G}' = \mathcal{G} \cup \{\langle pos, n \rangle\}$ contains the original goals and requires the *pos* variable to be at the end of the plan (this could be omitted but it can be useful for heuristics).

Plans for Π^{skip} only contain $skip$ actions (with a corresponding skipped action in the original plan) and actions from the original plan in the same order (if they appear in the plan at position i they are only applicable when *pos* is $i-1$). Therefore, there is a one-to-one correspondence between the actions in a plan π' for Π^{skip} and the actions in the plan π for Π, defined by the action positions. Consequently, it is straightforward to transform a plan for Π^{skip} into a plan for Π, i.e., by removing skip actions and replacing the remaining actions by their counterparts in Π.

The plan obtained from solving Π^{skip} can contain redundant zero-cost actions. In terms of plan cost, these type of redundant actions do not have any negative effects, but if the goal is to find plans without redundant actions, costs from the original task must be adapted. We adapt the original costs of the input plan actions by setting the cost of all zero-cost actions to 1, and multiplying all other costs by the factor $f = \lceil \frac{m}{\mathit{mincost}} + \epsilon \rceil$, where m is the number of zero-cost actions in the input plan, *mincost* is the smallest positive action cost in the plan, and ϵ is an arbitrarily small positive real number. (If $\mathit{mincost} = 0$, f is undefined but also unneeded.) This factor satisfies $f \cdot m < \mathit{mincost}$, which guarantees that removing any action with a cost greater than zero will be more beneficial than removing any set of zero-cost actions. With this modification, an optimal plan for Π^{skip} is a MR for π.

Enhancing Π^{skip}

Identifying if *all* subsequences of actions in a plan are necessary is NP-hard (Fink and Yang 1992; Nakhost and Müller 2010). However, this does not mean that we cannot identify *some* actions as necessary in polynomial time. Med and Chrpa (2022) defined *plan action landmarks* as actions that must be part of any plan reduction of a given plan. For example, if only a single action a achieves a goal fact, removing a would render the plan invalid. Furthermore, if some preconditions of a are also achieved by only one action a', then a' is also necessary. We call this specific type of plan action landmarks *trivial plan action landmarks* (TPAL). To simplify the formal definition of trivial plan action landmarks, we extend a given plan with *virtual initial* and *goal* actions, defined as $a_0 = \langle \emptyset, \mathcal{I} \rangle$ and $a_{n+1} = \langle \mathcal{G}, \emptyset \rangle$, respectively.

Definition 3 (Trivial Plan Action Landmark, TPAL) *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ be a plan for Π extended with virtual initial and goal actions. Action a_i is a trivial plan action landmark iff: (1) $i = n+1$ (goal action) or (2) there is a trivial plan action landmark a_j, $i < j$, such that there is a fact $p \in \mathit{eff}(a_i) \cap pre(a_j)$, and there is no action a_k, $k < j$, $k \neq i$, such that $p \in \mathit{eff}(a_k)$.*

The set of facts in $\mathit{eff}(a_i)$ that comply with the condition specified in (2) are the reason why a_i is a TPAL. We will call this set the necessary effects, and will refer to it as $n\_\mathit{eff}(a_i)$.

The set of actions that can potentially be skipped is $A_s = \{a_i \in \pi \mid a_i \text{ is not a } \textit{TPAL}\}$. With this, we create an enhanced task $\Pi^{skip}_{TPAL} = \langle \mathcal{V}', \mathcal{A}', \mathcal{I}', \mathcal{G} \rangle$, where $\mathcal{V}', \mathcal{I}', \mathcal{G}$ are defined exactly as for Π^{skip}, but the set of actions $\mathcal{A}' = \{a'_i \mid 1 \leq i \leq n\} \cup \{skip_i \mid a_i \in A_s\}$ only has $skip_i$ actions for actions a_i that are not TPAL.

We can further extend the definition by including the effects of known plan action landmarks when identifying new plan action landmarks. We define this type of plan action landmarks as *fix-point plan action landmarks* (FPAL). Put simply, an action is a *FPAL* if it is the only achiever of a fact that is needed after another FPAL overwrote that fact.

Definition 4 (Fix-point Plan Action Landmark, FPAL) *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task and $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ be a plan for Π extended with virtual initial and goal actions. The action a_i is a fix-point plan action landmark iff: (1) a_i is a trivial plan action landmark or (2) there is a fix-point plan action landmark a_j, $i < j$, such that there is a fact $\langle v, d \rangle \in \textit{eff}(a_i) \cap \textit{pre}(a_j)$, and there is another fix-point plan action landmark a_k, $k < i$, with an effect $\langle v, d' \rangle \in \textit{eff}(a_k)$ where $d' \neq d$ and there is no other action a_l with $l \neq i$, $k < l < j$ such that $\langle v, d \rangle \in \textit{eff}(a_l)$.*

We can now use FPALs to only define skip actions for actions that are not FPALs.

Finally, we propose one final enhancement: encapsulate consecutive sequences of FPALs in a single macro action (Fikes, Hart, and Nilsson 1972). For two consecutive FPALs a_i and a_j, we build the macro a_{ij} as follows: $pre(a_{ij}) = pre(a_i) \cup (pre(a_j) \setminus \textit{eff}(a_i))$, and $\textit{eff}(a_{ij}) = \textit{eff}(a_j) \cup \textit{eff}|_{a_j}(a_i)$, where $\textit{eff}|_{a_j}(a_i)$ represents the effect of a_i restricted to those facts which variable does not appear in the effect of a_j: $\textit{eff}|_{a_j}(a_i) = \{\langle v, d \rangle \mid \langle v, d \rangle \in \textit{eff}(a_i) \wedge v \notin vars(\textit{eff}(a_j))\}$. Longer sequences of macros result from iteratively applying this definition. Soundness for macros of FPALs is guaranteed since they contain consecutive actions of a valid plan.

Integrating Minimal Reduction into FDSS

The only thing left to do, is to integrate the action elimination step into Fast Downward Stone Soup. We use Π^{skip} enhanced with FPALs and macro operators, and solve the resulting tasks using Fast Downward with A* and *hmax* (Bonet and Geffner 2001) as the heuristic function. The steps are straightforward: each time FDSS finds a new plan, we create and solve a Π^{skip} task. Then, we process the plan found for the Π^{skip} task to transform it into a plan for the original task. This transformation only consists of a few simple steps:

- Remove any *skip* actions that might be present.
- Break down macro actions of FPALs into individual actions.
- If cost scaling was done (i.e., if there were zero-cost operators in the input plan), compute the cost of the plan resulting from applying the last two steps using the costs of the original actions.

FDSS assigns a time limit to each planning algorithm, so we have to decide how much time to assign to the action elimination process. We went with the most simplistic approach and give the remaining time of the algorithm that found the plan to the action elimination procedure. We could have assigned a larger time limit to action elimination, but in most cases this process only needs less than a second to finish. For some particular domains where plans have thousands of actions and few to none actions are FPALs, the action elimination process might take a large amount of time to finish. We considered that reducing the time limit of subsequent planning algorithms in these cases was not worth the potential cost reduction that action elimination could achieve.

Conditional Effects

The base Π^{skip} compilation can be used in domains where actions may have conditional effects. However, the enhanced compilations with trivial and fix-point plan action landmarks need some modifications. In particular, how TPALs and FPALs are defined/identified must be 'tweaked'.

Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan that contains actions with conditional effects. For any action a, $cond(f)$ defines a potentially empty set of facts with the effect conditions for every $f \in \textit{eff}(a)$. Without conditional effects, if $a_i \in \pi$ is a TPAL, then there must exist at least one effect $f = \langle v, d \rangle$ such that $f \in \textit{eff}(a_i)$ and either $f \in \mathcal{G}$ or $f \in pre(a_j)$, for another TPAL a_j, $j > i$, with a_i being the only achiever of f either for $\mathcal{G}$ or for a_j. To extend this definition to account for conditional effects, if a_i was identified as a TPAL because of effect $f \in \textit{eff}(a_i)$, then f is a *necessary effect* of a_i and thus every effect condition $\langle v_c, d_c \rangle \in cond(f)$ has to be taken as a regular precondition to identify new TPALs with Definition 3. Otherwise, the effect's conditions in $cond(f)$ are not considered when identifying other TPALs.

Definition 5 (Extended PAL Precondition) *Let $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a planning task with conditional effects and $\pi = \langle a_0, a_1, \ldots, a_n, a_{n+1} \rangle$ be a plan for Π extended with virtual initial and goal actions. Given a PAL a_i, the extended PAL precondition of action a_i is $e\_pre(a) = pre(a_i) \cup \bigcup_{c \in n\_\textit{eff}(a_i)} cond(c)$, where $n\_\textit{eff}(a_i)$ is the set of necessary effects of a_i.*

When the input task has conditional effects, TPALs are identified using $e\_pre(a_j)$ instead of $pre(a_j)$ in condition (2) of Definition 3.

FPALs take into account the effects of other FPALs when identifying new FPALs. In similar fashion as what was explained for TPALs, Definition 4 can be modified in a way that, when identifying FPALs (condition 2 in Definition 4), an effect of $f \in \textit{eff}(a_k)$ is only considered if $cond(f) = \emptyset$ or f was the reason a_k was identified as an FPAL (i.e. it is a necessary effect).

Macro actions

In the presence of conditional effects, macro actions of consecutive FPALs constructed as described previously are no longer guaranteed to be sound. This happens because a sequence of FPALs might have been applicable in the original plan, but removing actions might change what conditional effects are applied when executing the actions composing the macro. To remedy this, we simply only create macros of consecutive FPALs that do not have conditional effects.

Limitations

Our action elimination module has two limitations: disjunctive preconditions and derived predicates.

The first limitation stems from two reasons: one is just a technical detail in our implementation, while the other comes from plan action landmarks, and this issue is shared between both limitations. First, the technical detail: when there are disjunctive preconditions, Fast Downward's translator module creates multiple actions with the same name but different preconditions: one action for each element in the disjunction. Our implementation of the action elimination module uses the actions' names as an *id*, causing it to behave inconsistently when there are multiple actions with the same name in the input task. If this detail in the implementation is changed, we could use the base Π^{skip} compilation to solve MR for tasks with disjunctive precondition and derived predicates. Regardless of this, the issue with plan action landmarks would remain. The definition of trivial and fix-point plan action landmarks would need to be tweaked, as well as the implementation to account for disjunctive conditions and derived predicates. Because of time constraints related to the submission, we did not make the necessary changes to use plan action landmarks in these cases.

Our experiments show that the use of plan action landmarks has a big impact on the time needed to solve the Π^{skip} tasks. Thus, we decided to not run action elimination in these cases, so Spock defaults to the vanilla behaviour of Fast Downward Stone Soup when there are disjunctive preconditions or derived predicates.

Acknowledgements

We thank the authors of the Fast Downward planning system and the authors of the Stone Soup 2018 portfolio for providing the foundation for our IPC 2023 submission.

This work has been partially funded by PID2021-127647NB-C21 project, MCIN/AEI/10.13039/501100011033/ and by "ERDF A way of making Europe". Also by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

Balyo, T.; Chrpa, L.; and Kilani, A. 2014. On different strategies for eliminating redundant actions from plans. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014))*, 10–18.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012a. Determining redundant actions in sequential plans. In *Proceedings of the 24th International Conference on Tools with Artificial Intelligence*, volume 1, 484–491. IEEE.

Chrpa, L.; McCluskey, T. L.; and Osborne, H. 2012b. Optimizing plans through analysis of action dependencies and independencies. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, 338–342.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial intelligence*, 3: 251–288.

Fink, E.; and Yang, Q. 1992. Formalizing plan justifications. In *Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14. Carnegie Mellon University.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, volume 2835.

Med, J.; and Chrpa, L. 2022. On Speeding Up Methods for Identifying Redundant Actions in Plans. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 252–260.

Nakhost, H.; and Müller, M. 2010. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 121–128. AAAI.

Seipp, J.; and Röger, G. 2018. Fast downward stone soup 2018. *IPC2018–Classical Tracks*, 72–74.