Exploiting N-gram Analysis to Predict Operator Sequences*

Christian Muise Sheila McIlraith Jorge A. Baier Michael Reimer
Department of Computer Science, University of Toronto, Toronto, Canada. M5S 3G4.
{cjmuise, sheila, jabaier, mreimer} @cs.toronto.edu

Abstract

N-gram analysis provides a means of probabilistically predicting the next item in a sequence. Due originally to Shannon, it has proven an effective technique for word prediction in natural language processing and for gene sequence analysis. In this paper, we investigate the utility of n-gram analysis in predicting operator sequences in plans. Given a set of sample plans, we perform n-gram analysis to predict the likelihood of subsequent operators, relative to a partial plan. We identify several ways in which this information might be integrated into a planner. In this paper, we investigate one of these directions in further detail. Preliminary results demonstrate the promise of n-gram analysis as a tool for improving planning performance.

Introduction

The augmentation of deterministic planners with a learning component has been a topic of much research in recent years. A learning component generally takes as input a set of high-quality problem- or domain-specific *bootstrap* plans from which plan properties are learned. These properties are then used to augment an existing planner or planning domain so that subsequent plan generation over related planning instances is improved. Improvement is generally measured by a reduction in plan generation time and/or by an increase in plan quality relative to the performance of the planner without the learning component input. A notable result of the interest in planning and learning was the creation of a new *Learning Track* at the 2008 International Planning Competition (IPC-2008). This competition attracted 13 entrants from 10 groups. Entries differed with respect to the types of knowledge learned and with respect to how this knowledge was incorporated into subsequent planning. Approaches included learning policies, macro operators, sub-goal decompositions, and improving value functions; the most successful all-round planner was simply a portfolio of existing planners.[1]

In this paper we take a different approach to learning that is inspired by research in computational linguistics, and in particular by the task of word prediction. In word prediction, given a partial sentence, the system predicts the next likely word to extend the sentence. Most of us have seen word prediction technology, which is common on mobile devices and often used by keyboard users with physical or cognitive disabilities. This predictive technology is often based on n-gram analysis (Garay-Vitoria and Abascal 2006).

N-gram analysis provides a means of probabilistically predicting the next item in a sequence. Due originally to Shannon, it has proven an effective technique for word prediction in natural language processing and for gene sequence analysis. Here, we investigate the utility of n-gram analysis in predicting operator sequences in plans. Given a set of sample plans, we perform n-gram analysis to predict the likelihood of subsequent operators, relative to a partial plan.

An *n*-gram is a subsequence of n items (e.g., letters, words, base pairs, plan operators) that can be found in a larger corpus of items. For example the subsequence of words "and success rate" is found quite commonly in the sequence of words that makes up this paper. n-grams are typically characterized in terms of the size of the subsequence. An n-gram of size 1 is a unigram, of size 2 is a bigram, and of size 3 is a trigram. N-grams can include "don't care" items. For example the subsequence "and", followed by any word, followed by "success rate", is a special form of n-gram sometimes referred to as a *skip-gram*. For the purposes of this paper, we refer to these "don't care" items as wildcards and refer to skip-grams and n-grams collectively as n-grams.

N-gram analysis results in statistics regarding the frequency of occurrence of n-grams within subsequences of a corpus. It takes as input a corpus together with the size of n-grams to be distinguished (n), and the size of the subsequence (window) within which an n-gram can be found. If the window size is larger than the size of the n-gram, then wildcards can be interspersed within the n-gram. Table 1 shows a subset of the information obtained by performing n-gram analysis on the text of this paper, with both n and the window size equalling 3. The three n-grams listed each occur 5 times within the paper.

Certainly the idea of using n-gram analysis seems intuitive and compelling, but there are a number of challenges to its use. What is the optimal window and n-gram size? What is the analogue of a word? Is it a lifted operator name or a propositional representation of a grounded oper-

*An abridged version of this paper appears in the Proceedings of the 19th International Conference on Planning and Scheduling (ICAPS-2009).

[1]Details can be found at:
http://eecs.oregonstate.edu/ipc-learn/

N-gram	*Freq. of occurrence*
and success rate	5
a set of	5
this paper we	5

Table 1: N-gram and occurrence frequencies found by n-gram analysis of the English text contained in this paper. A frequency measure of f for n-gram i means that i appears f times in the input text.

ator? If lifted, how are arguments encoded and how are the relationships between arguments in sequences of operators exploited? And how do we integrate this information into subsequent planning? We might, for example, enhance the quality of a search heuristics with our predictive information; we might use these statistics to define a preferred operator ordering for operator selection in enforced hill-climbing (EHC) or A* search; or we might use n-gram statistics to suggest macro operators that could be used to augment the set of operators used for planning.

In this paper, we elected to exploit lifted representations of operators for our original n-gram analysis, resulting in the determination of high frequency operator sequences and/or orderings. We then postprocessed our n-grams to discover argument relationships. The result is a set of high-occurrence *patterns*: a notion that refers to a sequence of operators plus zero or more equality constraints between the arguments of the operators. We use these patterns to match with the plan history during the EHC phase of FF (Hoffmann and Nebel 2001). By matching the patterns with the history and possible next operators, we construct a set of *familiar operators* to investigate that take precedence over the helpful actions generated by FF.

In the sections to follow, we describe our approach and present preliminary results that demonstrate the promise of n-gram analysis as a tool for improving planning performance. We conclude with some discussion and future work.

Approach

We divide the presentation of our approach into two parts. The first corresponds to the *learning phase*, in which we extract statistical information from a set of plans, and then compile this information into patterns that can be exploited by the planner. The second phase is the *planning phase*. Our planner is a modification of FF that exploits the learned patterns. Next we describe the two components.

Learning Phase

The learning phase of our approach involves a number of individual steps, which are described below.

1. Solve the *bootstrap* problem set. Problems are solved using the Lama planner (Richter, Helmert, and Westphal 2008), run until it produces the best possible solution.[2]
2. Convert each of the discovered plans into sequences of operator names.

[2]Lama is often able to prove optimality with problems of this size.

```
768
move<>move<>221 462 417
move<>detonate-bomb<>78 462 96
pickup-bomb<>move<>73 96 417
```

Table 2: Example output of the n-gram statistical package on the benchmark plan solutions. The first line indicates the number of n-grams found. Each following line describes an n-gram followed by the occurrence frequency and two other statistics that are not used in this paper. The string <> indicates that the previous word is treated as a single token.

3. Run n-gram analysis on the operator-name sequences. Here we run a standard n-gram statistics package (Banerjee and Pedersen 2003) on the sequences of operator names with various n-gram sizes and window sizes. In our experiments, the sizes we chose to investigate range from 2 to 5 and the window sizes range from 2 to 10. Example output from the n-gram analysis can be seen in Table 2.
4. Construct patterns and pattern scores for each of the top n-grams. Patterns are the central piece of information that is exploited by the planner. Intuitively, they describe families of sequences of operators and corresponding argument constraints that seem to appear very frequently in solutions. Further elaboration is provided below.
5. Determine the best *settings* for our planner. A setting is a collection of patterns together with a parameter that estimates the number of *familiar operators* the planner should use. These settings are produced by solving the *target* problems using our planner, and determining which *settings* are most effective in solving as many problems as possible. (Further details follow.)
6. Generate an experimental strategy. To maximize the number of problems solved, our planner will attempt to use not one, but possibly several of the best settings computed in the previous step. During experimental evaluation of our planner, each setting is tried for 120 seconds and then the planner switches to the next best setting. The order in which the settings are tried is computed with a greedy algorithm that attempts to maximize coverage.

Constructing Patterns After retrieving the n-gram statistics, we must compile the information into a form that can be used within a planner. Using only the operator sequences, however, is not enough. To illustrate this point, consider the N-Puzzle domain where the only operator available is $move$. The n-grams generated from a corpus where only a single operator appears (i.e. $move$) provide little value. To extract more information from the generated plans, we consider the relationship arguments have inside of the n-gram.

For each of the most frequent n-grams, we find all of their occurrences in the generated plans and generalize the grounded arguments. For example, consider the n-gram $\langle move, move \rangle$ in a planning domain where movement can occur between locations connected to one another. When

examining the previous plans, we may find an occurrence of the following which matches the n-gram:

$$move(loc_a, loc_b), move(loc_b, loc_c)$$

By forgetting the operator names and listing the arguments from left to right, we can see that the second and third argument is the same ('loc_b'). Each set of argument indices is referred to as a *group*, and the set of argument groups, [1][2,3][4], along with the associated n-gram, constitutes a pattern. Each n-gram yields a set of patterns.

For use within the FF planner, we associate a score to each pattern. This score can then be used to choose which pattern to apply first in the generation of familiar operators. The scoring procedure for patterns consists of sorting every pattern, regardless of its associated n-gram, based on a generic comparing function. The score of a pattern is set to be its rank in this sorted list of all patterns. The comparing function we used is a series of tie-breaks on two patterns p_1 and p_2:

1. Prefer the pattern with higher frequency.
2. Tie-break: Prefer the pattern with more operators.
3. Tie-break: Prefer the pattern with fewer groups.
4. Tie-break: Prefer the pattern with more arguments.
5. Tie-break: Randomly pick one pattern.

Each subsequent line is only reached if p_1 and p_2 are equal on all previous lines. Intuitively, we want to give a higher score to more frequent patterns. If two patterns have equal frequencies, then we want the one with more grounded operators[3] since having more operators indicates more of the pattern has been matched. If these counts are equal as well, we choose the pattern with the smaller number of groups, as this captures more information about how the arguments relate to one another. Finally we select a pattern with the greatest number of arguments in general – if both patterns have the same number of groups, then more arguments indicates more information about argument relations.

This is not the only possible scoring function – one can imagine using a machine learning algorithm to discover a tailored scoring function on a per-domain basis. We hope to investigate options similar to this in future work.

Planning Phase: Using Patterns in FF

The automated planner FF (Hoffmann and Nebel 2001) operates in two distinct phases: EHC and best-first search. While there are a number of ways we could incorporate the use of patterns into FF, we decided to augment the set of successor states considered in the EHC stage. When a state is evaluated by the relaxed planning graph heuristic,[4] a number of helpful actions are generated for expanding the EHC search frontier. In many situations, however, this frontier is not sufficient to discover valid plans.

When expanding a state S, our planner takes the history of operators that lead to S and suggests *familiar operators* based on patterns with the highest score that match the history. Familiar operators play a similar role to helpful actions, and the number of familiar operators that should be added to the frontier is a run-time setting, and this value is considered on a per-domain basis to find the optimal bound. Having too many may cause the search frontier to become too large, yielding poor performance, so it is important to find the right trade-off. In general, we tested settings in the range of 1 to 15. Note that a setting of 0 causes FF to run without any change.

The familiar operators are a subset of all the applicable operators from the current state S. For each applicable operator A, we define $seq(A, S)$ as $history(S) \circ A$, i.e. the sequence of operators that correspond to the operators leading to S, plus the operator A. We then associate a score with A to be:

$$score(A) = \max_{p \in P_{seq(A,S)}} score(p),$$

where $P_{seq(A,S)}$ are the patterns that match the sequence $seq(A, S)$. If $|seq(A, S)|$ is smaller than the length of the pattern, it does not count as matching. Additionally, if $|seq(A, S)|$ is larger than the length k of a pattern, only the last k operators of $seq(A, S)$ are used for comparison.

Example To illustrate this procedure, consider the scenario in Figure 1 from the Gold Miner domain. Here, we are performing EHC and have arrived at state S after going through states 1, 2, and 3 via the operators $move$, $pickup\_laser$, and $move$. For the sake of this example, assume that neither $move$ nor *fire_laser* appear in the set of helpful actions, but are applicable in state S. Assume further that the following patterns exist:

$$\begin{aligned} p_1 &\equiv \langle pickup\_laser, *, \textit{fire\_laser} \rangle : [1, 2] \\ p_2 &\equiv \langle move, move \rangle : [1][2, 3][4] \\ p_3 &\equiv \langle move, *, *, move \rangle : [1][2][3][4] \end{aligned}$$

Pattern p_1 is of length three and has a single wildcard (any single operator matches this), p_2 is only of length two, and p_3 is of length four with two wildcards. The arguments for $pickup\_laser$ and *fire_laser* refer to the same object (since there is only one group), and the argument grouping for p_2 ensures that movement is through some specific location (the second group). For p_3, there is no relation between the two move operators. In the scenario described above, we have the following realization:[5]

- $history(S) = move, pickup\_laser, move$
- $seq(\textit{fire\_laser}, S) = move, pickup\_laser, move, \textit{fire\_laser}$
- $seq(move, S) = move, pickup\_laser, move, move$
- $P_{seq(\textit{fire\_laser},S)} = \{p_1\}$
- $P_{seq(move,S)} = \{p_2, p_3\}$
- $score(\textit{fire\_laser}) = score(p_1)$
- $score(move) = max\{score(p_2), score(p_3)\}$

Once the frontier has been constructed, the EHC will investigate the heuristic value of each state on the frontier and commit to one if it has a better value than S. The order in which these states should be investigated can have an impact

[3] Note that wildcard operators are not counted here.

[4] It is assumed that the reader is familiar with the relaxed planning graph heuristic, EHC, and FF in general.

[5] For ease of readability, we suppress listing arguments and assume the patterns match.

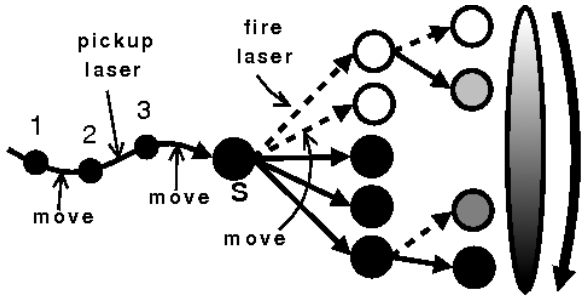

Figure 1: EHC Search in the Gold Miner domain. Solid lines after state S represent helpful actions and dashed lines represent familiar operators.

Domain	Time Metric		Quality Metric		Success Rate	
	FF	FF*	FF	FF*	FF	FF*
Gold Miner	0.001	19.19	6.86	30	0.26	1.0
Matching	0.13	0.11	7.42	7.41	0.33	0.33
N-Puzzle	0.22	0.03	10.16	8.29	0.53	0.43
Parking	0.03	0.03	4.04	4.04	0.23	0.23
Sokoban	0.14	7.41	10.22	11.83	0.43	0.43
Thoughtful	0.84	1.20	10.26	16.69	0.36	0.6
Overall	1.37	27.97	48.97	78.27	0.36	0.51
Delta	+ 26.60		+ 29.30		+ .15	
(O. Wedge)	(+ 36.05)		(+ 29.02)		(+ 0.17)	

Table 3: Time, Quality, and Success Rate Metrics of FF* and FF. For all metrics, larger numbers are better. Values for Obtuse Wedge were obtained using a slightly faster machine.

on the efficiency of the planner, so we chose to evaluate the familiar operators prior to the helpful actions.

If no state in the frontier is found to have a better heuristic value, the search continues to a deeper level. It is interesting to note that states in a level deeper than two may have been reached from applications of both familiar operators and helpful actions. Because of the ordering we place on frontier evaluation, at level k in the breadth first search we evaluate states from the most familiar (having been reached by applying familiar operators only) to the most helpful (having been reached by applying helpful actions only). Figure 1 demonstrates the spectrum of states that are investigated with lighter shades representing familiar operators and darker shades representing helpful actions.

Since we only modify the EHC phase of FF, if it fails to find a solution in this phase our approach behaves exactly as FF normally would.

Experimental Results

To evaluate our approach, we reproduced the experimental framework used in the IPC-2008 Learning Track. We built a testing framework analogous to the learning track, with the same problem sets used during the contest – 30 bootstrap / 30 target problem for training, and 30 target problems for testing. FF was run with and without the learned patterns (FF with the pattern usage enabled is referred to as FF*). Three primary metrics are associated with each planner in the contest: time, quality (plan length), and success rate. For each problem p, the best time and quality found by any planner is referred to as T_p^* and N_p^* respectively. The performance of any particular planner on p is denoted as T_p and N_p, and its score for the problem is calculated to be T_p^*/T_p and N_p^*/N_p. The overall time and quality metric for a planner is the sum over all of the individual problem scores.[6] Finally, the success rate is the percentage of problems solved in the given time limit of 15 minutes per problem. All experiments were conducted on a Linux desktop with a Dual Core 2.13GHz processor and 2GB of memory.

Unfortunately, most of the planners that were entered in this track are not publicly available. To compute our time and quality metrics, we compared to the raw data generated by the planners in the competition, giving us a lower bound on the score since the hardware configuration we used was slower than the contest machines.

Time, quality, and success rate metrics for the six domains used in the Learning Track are given in Table 3. For one domain, Parking, we were unable to learn any patterns since every configuration tested by the automated process caused the planner to fail in the EHC phase – thus results shown are for vanilla FF (the fall-back solution to our approach).

FF* displayed impressive performance along some dimensions and only modest performance along others. Most impressive are our Delta values – values that measure the effect of learning on the performance of our planner. Table 3 lists our Delta values for time, quality and success rate, as well as those of Obtuse Wedge (Yoon, Fern, and Givan 2008), recipient of the *Best Learner Award*. Delta values indicate that our planner displayed competitive results to Obtuse Wedge, even in the limited setting of slightly slower hardware (the competition machine ran at 2.4GHz). Our approach to learning is significantly different than that of Obtuse Wedge. We conjecture that combining both techniques may provide even better results.

The results for some domains are quite sensitive to the runtime. For example, in the Gold-miner domain FF* averaged about 0.06 seconds per problem in the target set, while the fastest scores in the contest averaged around 0.04 seconds. This difference of about two one-hundredths of a second cost FF* roughly 10 points in the time quality metric – a difference that would bring the delta score for time beyond that of Obtuse Wedge. This is simply a consequence of the testing framework of the IPC-2008 Learning Track on domains where the problems are solved very quickly.

For readability, we do not include all of the settings learned for each of the domains. However, some trends are interesting to note. For the domains where our FF* approach performs well, a large number of patterns were included from the learning phase ($\sim$70 on average). In the domains where FF* performed poorly, the bootstrap plans that were used as input to the learning phase did not result in a large number of high frequency n-grams ($\sim$5 on average). Conse-

[6] Note that the maximum score a planner can have for a particular problem is 1.

quently, few patterns were generated and little information was provided to the planner during the evaluation phase.

The size of n-gram used and the number of familiar operators varied from domain to domain and no clear trend for these settings emerged. For example, in the case of the Gold Miner domain n-grams of size 3 or 4 with a window size of 5 or 6 proved most useful. In the case of the Thoughtful domain n-grams of size 2 with a window of size 3 was the most successful.

Further analysis of the IPC-2008 statistics indicates that while most planners' learning components negatively impacted quality, ours had a positive impact, showing improvement in both quality and success rate metrics. These observations lead us to conjecture that perhaps the other planners' learning components generated too much overhead, resulting in an inability to solve problems within the allotted time. FF* did not suffer this fate.

Our planner did have its shortcomings. In general FF* scored in the middle of the pack, or slightly better relative to the IPC-2008 planners for time, quality, and success rate. It excelled in a few domains, but was unexceptional in others. Nevertheless, the preliminary nature of this work and the fact that our Delta values characterize us as one of the top learners relative to all other IPC-2008 entrants does much to demonstrate the promise of this line of research.

Future Work

The interpretation of serialized plans as sentences in a language reveals a number of interesting avenues to investigate. We made the assumption that the 'sentences' are formed only from the operator names. This can be generalized to include other aspects of a planning problem such as state, goals, and landmarks. Using a richer source of information as input to the n-gram analysis may lead to better results.

Another way in which we can augment the pattern is to extend the concept of a wildcard to resemble the Kleene star in regular expressions. This would allow us to perform a sequence of operators of the same type (albeit with different sets of arguments) before continuing the pattern. For example in a delivery type domain, a pattern could describe the concept of $pick\_up$, then $move$ an arbitrary number of times, and then put_down. This additional power could drastically improve the effectiveness of patterns in some domains.

The manner in which we use the learned information is another aspect we hope to develop further. Currently, only the EHC phase of FF benefits from the pattern knowledge, but the second phase of Best First search can be modified as well. Similar to the reactive policies described in (Yoon, Fern, and Givan 2008), a plan can be rolled out to a certain depth using the highest scoring familiar operator. All of the states along this path can be added to the Best First search neighbourhood, and this would allow both phases of FF to benefit from the learned pattern knowledge.

Another compelling idea we hope to investigate is the relationship between our approach and macro operators. In contrast to macro operator approaches, our method matches with the pattern history applying only one operator to obtain a successor. An investigation of the use of patterns in a way more similar to a macro operator – in which we would apply more than one operator to generate a successor – is an interesting avenue of future research.

The applicability of our approach need not be confined to IPC planning domains. For example, further extensions may be realized that allow a life-long robot companion (e.g. Thrun and Mitchell 1995) to learn habitual patterns in every day routines – even allowing those patterns to change over time; given sufficient background behaviour, plans being executed can be monitored for any discrepancies observed as statistical anomalies compared to the known n-gram statistics. Many aspects of plan monitoring could similarly benefit from the use of n-gram analysis on operator sequences.

Conclusions

In this paper we presented a new perspective on planning strategies through the lens of computational linguistics. By treating partial plans as incomplete sentences, we showed how word prediction techniques can help discover effective operators during forward-search planning. We introduced the concept of a *pattern* which extends the notion of an n-gram to include equivalence constraints amongst arguments of the operators. With the n-gram of operator names, and argument restrictions of a pattern, we suggest a set of *familiar* operators for the EHC search to choose from.

We evaluated our method with respect to the IPC-2008 Learning Track framework. Preliminary results are encouraging, and indicate that this approach is a powerful learning technique.

As with most learning algorithms, having a larger corpus for training can have a significant impact on the quality of statistics. It is likely that with increased training, further patterns and improved frequency estimates would arise from the use of n-gram analysis.

Acknowledgements

The authors gratefully acknowledge funding from the Ontario Mininistry of Innovation for support through their Early Researcher Award (ERA) and the Natural Sciences and Engineering Research Council of Canada (NSERC) for support through their Discovery Grant and Graduate Scholarship programs. We would also like to thank the anonymous referees for useful feedback on earlier drafts of the paper.

References

Banerjee, S., and Pedersen, T. 2003. The Design, Implementation and Use of the Ngram Statistics Package. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, 370–381.

Garay-Vitoria, N., and Abascal, J. 2006. Text prediction systems: a survey. *Univers. Access Inf. Soc.* 4(3):188–203.

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

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

Thrun, S., and Mitchell, T. 1995. Lifelong robot learning. *Robotics and autonomous systems* 15(1):25–46.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning Control Knowledge for Forward Search Planning. *JAIR* 9:683–718.