A quick policy to filter reactions based on feasibility in AI-guided retrosynthetic planning

Samuel Genheden*, Ola Engkvist, Esben Bjerrum

Molecular AI, Discovery Sciences, R&D, AstraZeneca Gothenburg, Sweden

* Corresponding author: samuel.genheden@astrazeneca.com

Abstract

A neural network-guided Monte Carlo tree search (MCTS) has been shown to be a promising algorithm for computer-aided synthesis prediction. Here we train and investigate a filter policy that removes unfeasible reactions from the search. We investigate three different methods to generate negative data that the filter policy model can be trained on, and we show that these methods are complementary to each other. Therefore the most robust model is one that combines all generated negative data. The filter in itself is quick (< 0.1 s per prediction on average) but using the filter policy in the MCTS results in a doubling of the search time. However it only leads to a small reduction in the success rate of finding synthetic routes (< 1%) and we are able show that using filter policy the MCTS algorithm produce more promising routes, although the predicted routes are more complex. The filter policy has been integrated in the AiZynthFinder software.

Keywords: neural networks, Monte Carlo tree search, synthesis prediction, reaction feasibility

Introduction

Computer-assisted synthesis prediction[1,2] has undergone a tremendous change the last couple of years from hand-encoded rules based on a rather small database of reactions[3,4] to automated algorithms guided by artificial intelligence that has been trained on millions of known reactions.[5,6,7] The seminal paper by Segler et al.[8] showed that suitable retrosynthetic templates for a compound[9] could be coupled with a Monte Carlo tree search (MCTS)[10] in an efficient retrosynthetic planning tool. Similar ideas forms the basis of the open-source ASKCOS suite of tools.[11] Thakkar et al. implemented a simplified version of the MCTS algorithm and investigated the effect of the dataset used to train the neural network.[12] The code used by Thakkar et al., was recently productionized and released to the public as the AiZynthFinder tool.[13]

The MCTS algorithm consists of four steps: selection, expansion, rollout and update.[9,10] In the expansion phase new nodes are added to the search tree: first, the precursor molecules represented by the node which is to be expanded are subjected to an expansion policy, a neural network trained to recommend retrosynthetic templates for a given compound. Typically, only the top-50 templates are returned to limit the search space, because it is not likely that the remaining templates are applicable on the query molecule. The second step is to apply the templates *in-silico* on the query molecules, which will produce the new nodes in the search tree. The edge between the parent and child node can be seen to represent a chemical reaction in the backwards direction. However, there is nothing inherent in this produce that guarantees that the chemical reaction is feasible, i.e. it would be a successful reaction in the wet-lab. The only guarantee is that the template is applicable *in-silico*. To improve the accuracy of the MCTS algorithm, Segler et al., introduced an in-scope filter; [9] in a third step

of the expansion phase, the created reactions are subjected to a filter policy, a neural network trained to determine if a reaction is feasible or not. Only reactions and the newly created child nodes, that are judged to be feasible, are kept. To train the filter policy, Segler et al., constructed artificially unfeasible reactions using two strategies. First, they created 30 M unfeasible reactions by taking the additional products formed when applying the recorded templates to the reactants due to lack of template specificity. Secondly, they created 70 M unfeasible reaction by randomly picking a template for a random set of reactants.

In this paper we investigate the choice of method for generating negative data, its influence on the filter policy and consequences for the route finding in the MCTS algorithm as implemented in the AiZynthFinder software.[13] We build upon the existing methods for generating negative data,[8,14] but further suggests a third and novel method designed to give more plausible negative examples that are more difficult to distinguish from the reactions recorded in literature. We investigate and compare the three approaches to artificially generate unfeasible reactions on their ability to generalize between the constructed datasets. Furthermore we study the effect of the resulting filter policies on the MCTS algorithm ability to find plausible synthetic routes for a given molecule Our main objective is not to find as many possible routes as possible, but routes that seem plausible to an expert chemists. This is a better metric than simply the number of solved routes as this might hide unfeasible routes.

Methods

Template library. The database of reaction templates was obtained from the publicly available US Patent Office dataset[15] as prepared by Thakkar et al.[12] The prepared library consists of 911,869 reactions distributed over 46,695 unique templates.

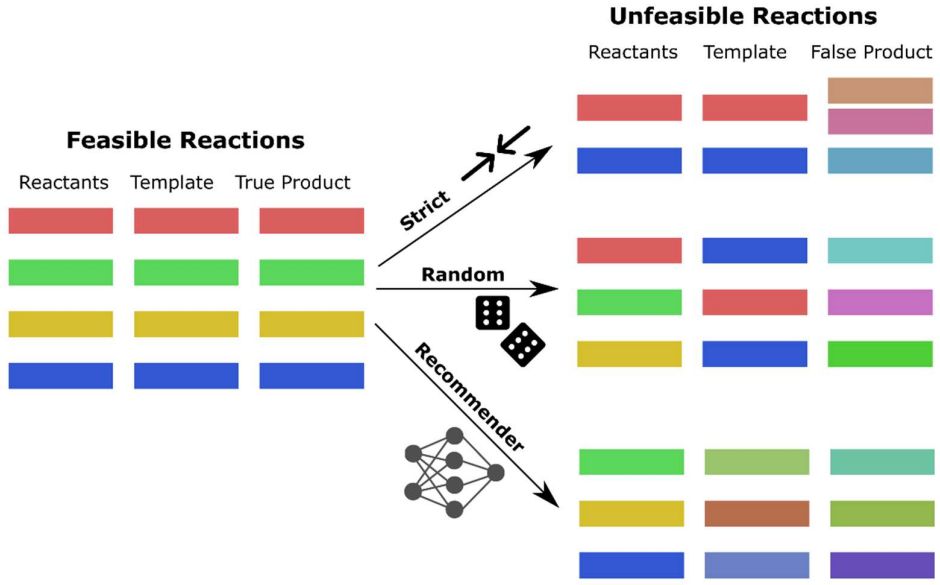


Figure 1 – Illustration of the approach to generate negative data, i.e. unfeasible reactions. The template library with feasible reactions are shown to left and consist of rows of reactants, templates and true products (in the illustration there a four such rows). The different approaches to generate unfeasible reactions then pick a template row for each row of reactants which gives false products. Not all rows of reactants gives false products.

Generation of negative data. In order to generate negative data, i.e. reactions that are unlikely to be successful in the wet-laboratory, three different procedures were employed (see Figure 1). In the first, which we will denote as *strict* in the following, we applied the exact template recorded for each row of reactants in the template library. This gives the recorded product but also other products due to the non-specificity of the template. The additional products together with the recorded reactants were taken as negative data. In the second procedure, which we will denote as *random*, we picked a random template from the template library for each row of reactants in the dataset. Templates that were identical to the dataset template, as well as templates that produced the recorded product, were discarded. 1000 trials were attempted for each row of reactants; if an applicable template could not be found after these many iterations, the search was terminated. In the third procedure, which we will denote as *recommender*, we trained a neural network to suggest templates. We looked among the top-20 suggested templates for applicable templates, and as with the *random* procedure, we discarded the recorded template or any templates that gave the recorded product. The recommender neural network is described next.

Template recommender neural network. The architecture of this neural network follows the architecture of the rollout policy used in the tree search algorithm, [12] but adapted to use on reactant molecules as follows. The input is a 2048-bit fingerprint (ECFP4, computed by the Morgan algorithm in RDKit[16,17]) of the reactants that were created by summing the fingerprints of the individual reactants. The input is passed to a fully connected layer with ELU activation followed by a drop-out layer and finally a SoftMax activation to give probabilities for each of the templates in the library. In the model used to generate negative data, we used 512 nodes in the dense layer and a drop-out rate of 0.4, although we attempted a limited hyper-parameter search with 1024 nodes in the dense layer and drop-out rates of 0.5 and 0.6. The accuracy of the validation set was not significantly affected by the exact values of the hyper-parameters. We trained the neural network using the USPTO template library.[15] Templates were represented as binarized labels in a one-vs-all fashion. Training, validation and test sets were constructed as random 90/5/5 split of the template library. We used the Keras interface of Tensorflow[18] employing an Adam optimizer with an initial learning rate of 0.001, and categorical cross entropy loss. The training was carried out for 50 epochs with a batch size of 256.

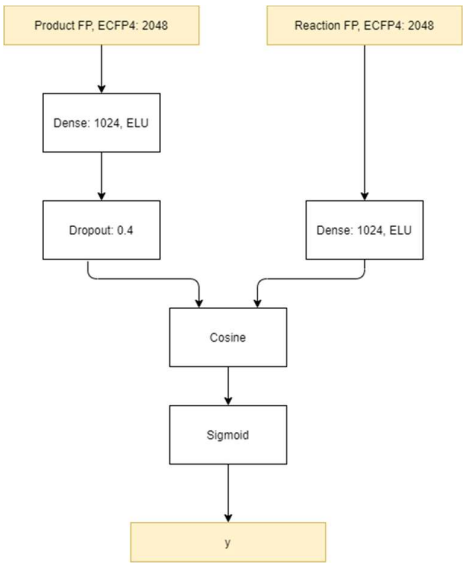


Figure 2 – Network architecture for filter policy

Filter policy training. The architecture of the neural network for the filter policy closely follows the architecture outlined by Segler et al.[8] and is depicted in Figure 2. The network has two inputs: the ECFP4 of the product and the ECFP4 of the reactions, constructed as a difference fingerprint (the sum of the fingerprints of the reactants are subtracted from the fingerprint of the product).[19] The output is the probability of the feasibility of the reaction. The model was trained by concatenating the USPTO template library with the negative data, the output being 1 if the reaction was from the template library and 0 if it was from the negative dataset. Training, validation and test sets were constructed as random 90/5/5 split of the template library. We used the Keras interface of Tensorflow[18] employing an Adam optimizer with an initial learning rate of 0.001, and binary cross entropy loss. The training was carried out for 50 epochs with a batch size of 256.

Evaluation of filter policies. To evaluate the performance of the filter policies on the test set we use a selection of metrics. Balanced accuracy is the average of the recalls of the feasible and unfeasible reactions,[20] and is used instead of plain accuracy because some of the test sets are highly imbalanced. Average precision or the area under the curve of the precision-recall graph is used to gauge the performance over the whole range of decision cutoffs. Recall is the fraction of feasible reactions predicted to be feasible by the filter. The uncertainty of the balanced accuracy, average precision and recall was calculated by bootstrapping, using 500 resamples.

Route finding. The filter policy was implemented in the AiZynthFinder software.[13] The filter was applied when a child node is selected for the first time; the reaction leading to this node is evaluated by the filter policy and if the probability returned by the policy is lower than a threshold the child node is discarded. The threshold depends on the filter policy and is discussed below. To evaluate the effect of the filter on the capability of the tree to find routes, we created a set of compounds of 5000 random molecules from ChEMBL.[21] The compounds were assigned the most likely tautomer using RDKit.[17] The SMILES strings[22] of these compounds were given as input to the AiZynthFinder software, using an internal stock of molecules as stop condition.

Results and discussion

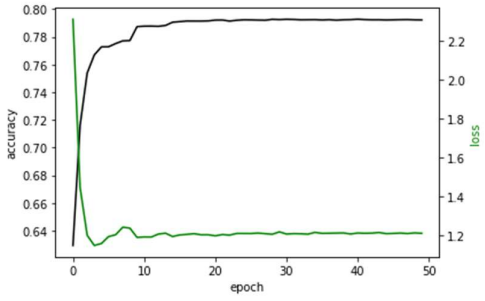


Figure 3 – Validation loss and accuracy as a function of epoch for the training of the recommender neural network

Evaluation of the recommender neural network. The training of the neural network model for the *recommender* converges rather quickly as seen in Figure 3. The accuracy of the validation set converges to 0.79 within circa 15 epochs. The top-10, top-20 and top-50 accuracy of the validation set converges to 0.95, 0.96 and 0.97, respectively, and the same statistics are obtained for the test set as well. Because the negative templates for each reactant will be drawn among the top-20 suggested ones,

this level of accuracy seems appropriate for the intended application. Therefore, no further optimization of the recommender network was attempted.

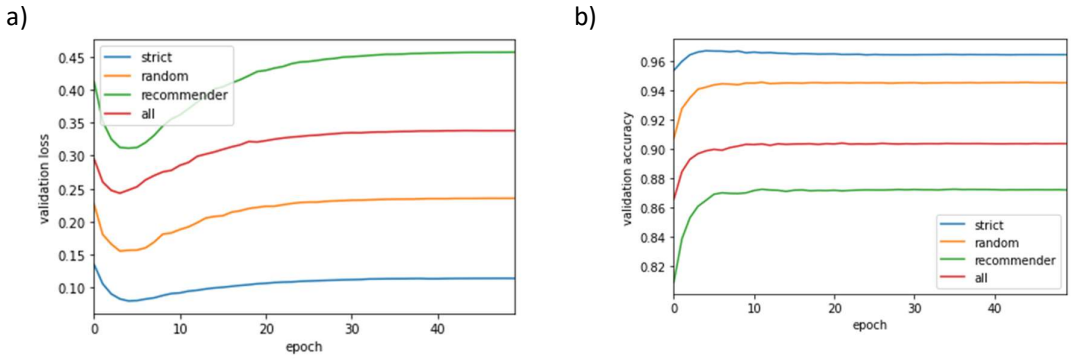


Figure 4 – Validation loss (a) and validation accuracy (b) as a function of epoch for four different policy filter models.

Table 1 – Statistics for different filter policy model[1]

Model	Number of negative data points[2]	Average precision	Balanced accuracy	Recall
strict	77,760 (7.9%)	1.00	0.88	0.98
random	838,057 (47.9%)	0.99	0.95	0.97
recommender	705,667 (43.6%)	0.96	0.87	0.90
strict + random	915,817 (50.1%)	0.98	0.93	0.95
strict + recommender	783,427 (46.2%)	0.95	0.86	0.88
random + recommender	1,543,724 (62.9%)	0.95	0.91	0.89
all	1,621,484 (64.0%)	0.95	0.90	0.89

[1]The balanced accuracy and the recall was determined at a decision cutoff of 0.5. The bootstrapped uncertainty of the average precision, balanced accuracy, and recall is less than 0.005 for all sets. [2]The number in parenthesis show the percentage of the number of negative data points to the total number of data points.

Table 2 – Balanced accuracy for different combinations of model and test dataset[1]

	Filter policy model						
Test set	strict	random	recommender	strict + random	strict + recommender	random + recommender	all
strict	**0.88**	0.67	0.73	0.89	0.88	0.76	0.88
random	0.59	**0.95**	0.95	0.98	0.94	0.98	0.98
recommender	0.53	0.79	**0.87**	0.80	0.95	0.97	0.97
random + strict	0.62	0.96	0.93	**0.93**	0.94	0.96	0.98
strict + recommender	0.57	0.78	0.94	0.81	**0.86**	0.95	0.96
random + recommender	0.57	0.90	0.96	0.90	0.96	**0.91**	0.96
all	0.58	0.89	0.95	0.90	0.96	0.95	**0.90**

[1] The numbers in bold on the diagonal are the same numbers as in Table 1.

Performance of filter policy models. We trained several filter policy models based on different collections of negative data: Apart from training models based on just the *strict*, *random* and *recommender* datasets, we also trained on combinations of these sets. The positive dataset was always the USPTO dataset. A shown in Table 1, the number of negative data points ranges from 77,760 for the *strict* set to 1,621,484 for the combination of all three datasets. The proportion of negative data points in the s*trict*, *random* and *recommender* sets are 7.9, 47.9 and 43.6%, respectively, showing a spectrum

of imbalanced datasets. The training for all models converges quickly as seen in Figure 4, the validation loss is converged after approximately 30 epochs and the validation accuracy after only approximately 15 epochs. Even though the validation loss show a minimum, the validation accuracy is not affected. We attribute this behavior to that the network predicts more towards 0 or 1 as the training progresses. This affects the cross entropy loss, but not the accuracy. The balanced accuracy, the average precision and the recall is shown in Table 1. Overall the performance is excellent as judged by the average precision, which ranges from 0.95 to 1.0. The average precision should be a better metric for the overall performance than the ubiquitous area under the curve of the ROC curve for imbalanced dataset, because it is sensitive to the sizes of the different classes. Furthermore, the recall is excellent for the *strict* and *random* models, but less so for the combined models. However, the optimal decision threshold is likely not 0.5 for all models. The balanced accuracy ranges from 0.86 for the *strict + recommender* combinations, which is reasonable, to 0.95 for the *random* model, which is excellent.

Next, we used the balanced accuracy to compare the performance of the models on different test datasets, which may thus contain negative samples derived by other means than used for the training set. This is shown in Table 2. For the *strict* model, we see that it only performs well on its own test set, whereas the *random* and *recommender* models perform well on many test sets but worst on the *strict* dataset. The *recommender* model performs better on the *random* test set, than it does on its own test set, whereas the opposite is true for the *random* model. This fits our hypothesis and intention that the recommender dataset on average should contain harder to predict negative examples than the random set. Filter policies trained on the random set have not been presented with so many of the hard examples during training and more often fails to predict them correctly. Using the top-20 for the recommender set may on the other hand still include some samples which are like the random after the first few suggestions. It may be likely that choosing another threshold for the recommender network data could influence these results. The lower performance of all the different single train set policies while tested on differently derived data, highlights that the three different procedures to generate negative data complements each other. Furthermore, performance increases are often seen with combinations of datasets are used for training. We see that any combination model including the *strict* negative dataset performs well on all test sets, and often combinations of trainsets increases the performance in comparison with the test set derived with the same method as the training set. As example the balanced accuracy is improved from 0.87 to 0.95 and 0.97 for the recommender test set by adding strict or random data during data. This cannot be due to the increased train set size alone, as the strict dataset is by far the smallest (c.f. Table 1). The best overall performance is, perhaps unsurprisingly, obtained by combining all three datasets of negative data during training. Therefore, we will continue evaluating the performance of the retrosynthesis tool using four sets of filter policies: *strict*, *random, recommender* and *all.*

Table 3 – Recall and specificity for different filter policy models at different levels of decision cutoff

	Strict		Random		Recommender		All	
Threshold	Recall	Specificity	Recall	Specificity	Recall	Specificity	Recall	Specificity
0.05	0.99	0.65	0.99	0.88	0.96	0.72	**0.98**	**0.85**
0.10	0.99	0.68	0.98	0.89	0.94	0.75	0.98	0.87
0.15	0.99	0.70	0.98	0.90	0.93	0.77	0.97	0.88
0.20	0.99	0.71	0.98	0.91	0.93	0.79	0.97	0.89
0.25	0.99	0.72	**0.98**	**0.91**	0.92	0.80	0.97	0.89
0.30	0.98	0.73	0.98	0.91	0.92	0.81	0.96	0.89
0.35	0.98	0.74	0.97	0.92	**0.91**	**0.82**	0.96	0.90
0.40	0.98	0.75	0.97	0.92	0.91	0.83	0.96	0.90
0.45	0.98	0.76	0.97	0.92	0.90	0.84	0.95	0.90
0.50	0.98	0.77	0.97	0.93	0.90	0.84	0.95	0.91
0.55	0.98	0.78	0.97	0.93	0.89	0.85	0.95	0.91
0.60	0.98	0.79	0.97	0.93	0.88	0.86	0.95	0.92
0.65	**0.98**	**0.80**	0.96	0.93	0.88	0.87	0.94	0.92
0.70	0.98	0.82	0.96	0.94	0.87	0.88	0.94	0.92
0.75	0.98	0.83	0.96	0.94	0.86	0.88	0.94	0.93
0.80	0.98	0.84	0.95	0.94	0.85	0.89	0.93	0.93
0.85	0.97	0.86	0.95	0.95	0.84	0.90	0.93	0.93
0.90	0.97	0.87	0.94	0.95	0.82	0.91	0.92	0.94
0.95	0.97	0.90	0.93	0.96	0.79	0.93	0.90	0.95

Numbers in grey are outside our chosen thresholds and numbers in bold highlights the chosen threshold

Decision cutoff. In the retrosynthesis tree search, we need to set a threshold to decide when to reject or to keep a reaction. We are mostly interested in the recall capacity (true positive rate), i.e. we do not want to reject feasible reactions. We still want to maintain a high specificity (true negative rate), i.e. we want to reject unfeasible reactions, although that is secondary to having a high recall. Therefore, we computed the recall and specificity for different thresholds and show them in Table 3. As guidelines we decided to have at least 0.90 recall and a specificity of 0.8. We also tried out a number of different thresholds and looked at distribution of the filter policies of the reactions explored in the tree search (see Figure S1) to determine that it was a smooth continuum of probabilities. For the *strict* model, we then need a threshold of 0.65, giving a recall of 0.98. For the *random* model, we could have a specificity of 0.9 and a recall at 0.98 if we choose a threshold of 0.25, so we used that. For the *recommender* model, we choose a threshold of 0.35, giving a recall of 0.92 and a specificity of 0.82. Finally, for the *all* model, we finally settled on a threshold of 0.05 giving a recall of 0.98 and a specificity of 0.85. We also tried a threshold of 0.35, but that rejected too many reactions in the tree search (see below).

Table 4 – Statistics of search time and route-finding capabilities

Filter	Average increase in search time	Number of solved compounds	Number of unsolved compounds
No	N/A (1)	3875	1125
Strict	2.0	3849	1151
Random	1.6	3863	1137
Recommender	2.0	3808	1192
All	1.6	3836	1164

Table 5 – Average difference in selected tree statistics between using a filter and not using one

Filter	Number of nodes	Number of steps	Number of precursors	Number of precursors in stock	Number of found routes
Strict	7.8	-0.04	-0.01	-0.01	5.7
Random	26.6	-0.03	-0.07	-0.07	15.9
Recommender	61.2	-0.10	-0.12	-0.10	30.5
All	30.4	-0.07	-0.10	-0.09	20.0

Effect of filter on tree search. We measure the effect of the filter policies on the tree search by the route finding capability. How much better or worse does the tree search become at finding a synthetic route for a target compound when a filter is applied? This is summarized in Table 4. First, we can see that the application of the filter policy roughly doubles the search time, and this is expected because the application of the filter policy increases the number of calls to the most expensive routines in the tree search: the application of the retrosynthetic template using RDChiral[23] and the predictions by a TensorFlow[18] neural network. The filter policy network is in itself fast, the average prediction time is 0.03 s for 4,000 random reactions from the search, but repeated calls to the network slows down the search. Furthermore, when using the filters, we also populate the search tree with many more nodes as seen in Table 5; the average number of nodes increases with up to 61 nodes. It is noticeable that whenever the *recommender* or *all* model is used, more nodes are populated. Furthermore, we see that there is a general decrease in the number of compounds for which the algorithm finds a synthetic route. The decrease ranges from 12 when using the *random* policy to 67 when using the *recommender* policy. (When the threshold of the *all* policy was set to 0.35, the number of unsolved compounds increased with 124). However, according to the statistics in Table 5, the number of steps and the number of precursors does not change substantially when using a filter. The reaction routes are slightly longer and therefore the number of precursors are higher. Finally, we can see that the number of found routes decreases by the application of filter.

Table 6 – Number of solved and unsolved compounds for different categories

	Solved *without* filter		Unsolved *without* filter	
Filter	Solved *with* filter	Unsolved *with* filter	Solved *with* filter	Unsolved *with* filter
Strict	3837	38	12	1113
Random	3800	75	63	1062
Recommender	3734	141	74	1051
All	3774	101	62	1063

Inspection of synthetic routes. It is of interest to dissect the differences between using a filter and not using one a little bit further by computing the contingency table for the four filter policies when comparing to using no filter. These counts are shown in Table 6. There are between 3734 and 3837 compounds for which a solution is found both when using a policy and when not using one, and there are between 1051 and 1113 compounds for which no solution is found irrespectively if a filter is used or not. The former compounds can be considered easy and unchallenging, whereas the latter compounds can probably be considered hard to predict with the current methodology. However, there are between 38 and 141 routes that are only found when not using a filter, and between 12 and 74 routes that are only found when using a filter, indicating that using and not using a filter can be slightly complementary.

As mentioned in the introduction, we aim to produce as many plausible routes as possible – not only as many routes as possible. Therefore, to analyze the quality of the predicted routes, we opted for analyzing and comparing the routes solved irrespectively if a filter or not was used. By comparing solved to solved routes, we can highlight different strategies used by the tree search when using or not using a filter. First, we analyzed the classes of the reaction used in the top-ranked routes. We extracted all the top-ranked routes for compounds for which a solution was found with all filters or with no filter, and then we listed all the reactions used in the top-ranked routes and what reaction class they belong to. For each unique reaction class we then computed the difference between the occurrence in routes produced when using no filter and the occurrence when using the *All* filter. The

top-5 occurrence differences by reaction classes with no filter and the *All* filter are shown in Table 7. We see that when a filter is used, more protective group chemistry is used as well as reactions that are unclassifiable. Furthermore, we see that the filter effectively lowers the usage of halogenation reactions and other functional group conversions. These kind of templates are likely to encounter selectivity issues, and hence it seems that the filter is especially good at picking up such situations.

Table 7 – Reaction classes and the difference in occurrence between using the *All* filter and not using a filter.

Reaction classes used more with *All* filter		Reaction classes used more when no filter is used	
Reaction class	Occurrence difference	Reaction class	Occurrence difference
NH deprotections	-497	Halogenation	996
RH deprotections	-373	Other functional group conversion	954
Unassigned	-238	N-acylation to amide	625
Heteroaryl N-alkylation	-109	Other functional group additional	341
O-acylation to ester	-107	Acid to acid chloride	152

We then extracted a number of solved routes that we inspected. We identified solved routes for which there was a large difference in the number of steps in the absence of filter and when using the *All* filter. In Figure 5, we show such a pair of routes that differ substantially in length. Without any filter, MCTS finds a route with a single step. However, this reaction is unlikely due to selectivity issues, and the *All* filter assigned a 0.005 probability to this reaction. When the *All* filter was used, the MCTS finds a route with six steps. The first reaction is a benzyl protection of one the hydroxyl groups, which appears as an attempt at resolving the selectivity issue, and is a more intelligent start of the route. However, the second step, an ester hydrolysis, is seemingly unnecessary as it is later reversed in step 5 . Still, over all the route suggested when using the *All* filter gives better options for a chemists to work with.

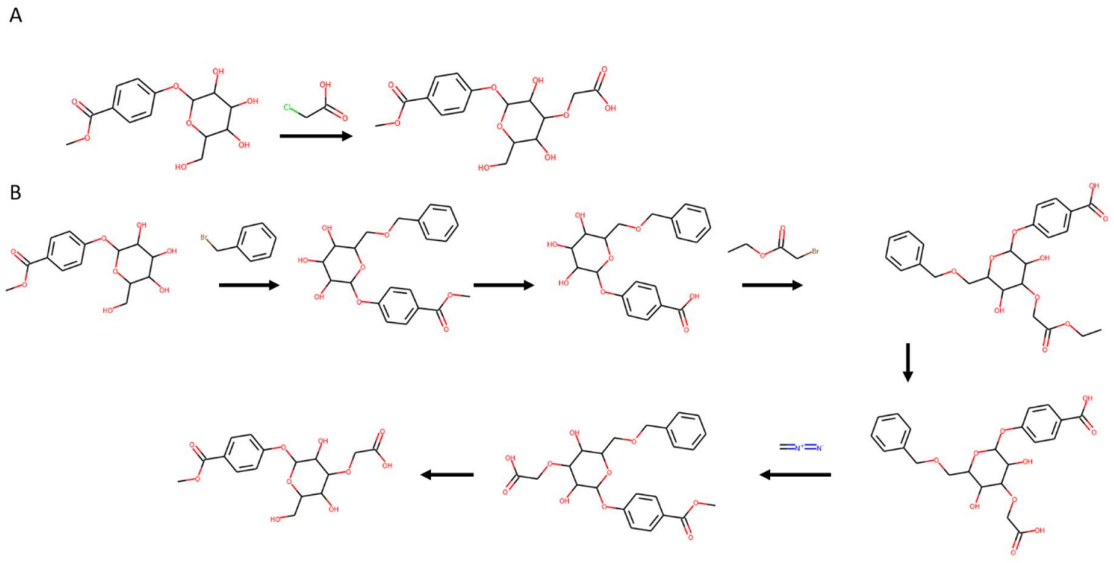


Figure 5. Example routes no 1. A) the route suggested when not using a filter, and B) the route suggested when using the *All* filter.

Another example is shown in Figure 6. Both of the routes involve similar reactions, and both end with an amide coupling. The route proposed when not using the filter has as the second step a C-O coupling with ethanolamine, but it is unclear if the reaction is sufficiently specific.[24] The *All* filter would have rejected this reaction with a 10^{-4} probability. The route proposed when using the *All* filter starts with a

benzylation, followed by bromination with N-bromosuccinimide and debenzylation. It then alkylates the hydroxyl group with a side chain which contains a highly sensitive azide group, before performing a Suzuki coupling. This route suggests a more elaborate sequence of chemistry than the shorter routes suggested when not using the filter. However, the order of the steps could be reshuffled or modified, to avoid the use of the azide group.

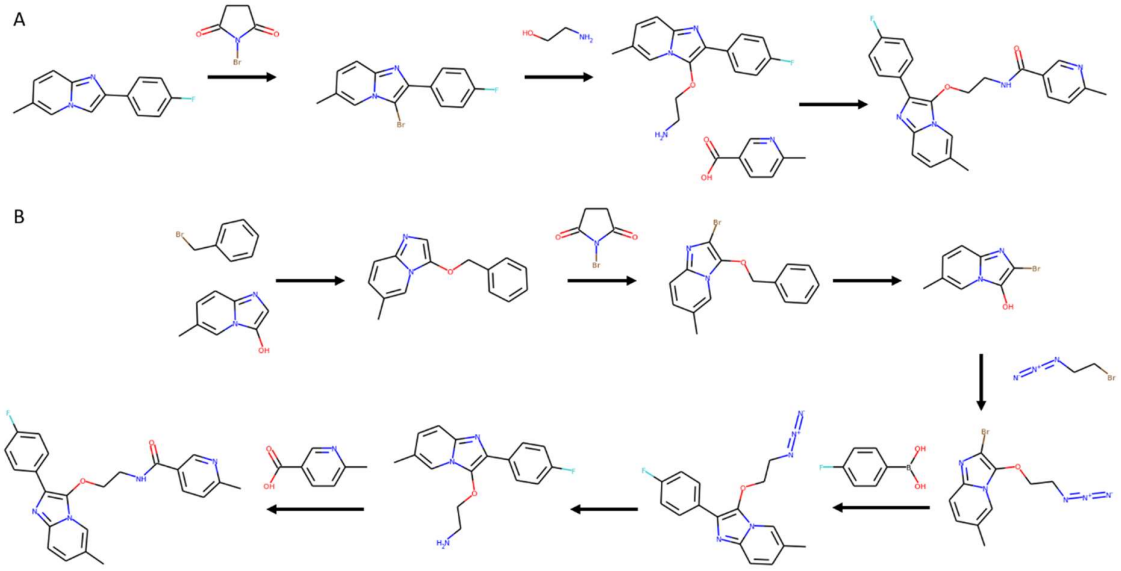


Figure 6. Example routes no 2. A) the route suggested when not using a filter, and B) the route suggested when using the *All* filter.

We also identify a few instances where MCTS with the *All* filter suggested a route that was shorter than the one suggested with a filter. An example of such a route is shown in Figure 7. For this compound, the route suggested when using no filter is questionable. For instance, the tosyl protecting group is removed on step 3 to then immediately replace it via reaction with benzenesulfonylchloride on step 4. On the contrary, the route suggested when using the *All* filter is shorter and convergent.

We show a few other such pairs of routes in the Supplementary information. In most cases, the route suggested when using the *All* filter consists of more complex chemistry than the route suggested when using no filter. The routes are not always perfect, and will require some optimization by a chemist. It is also not always clear how to interpret why the *All* filter rejects a reaction. However, we believe that the often longer routes suggested when using the *All* filter overall are a better source of ideas for synthesis planning.

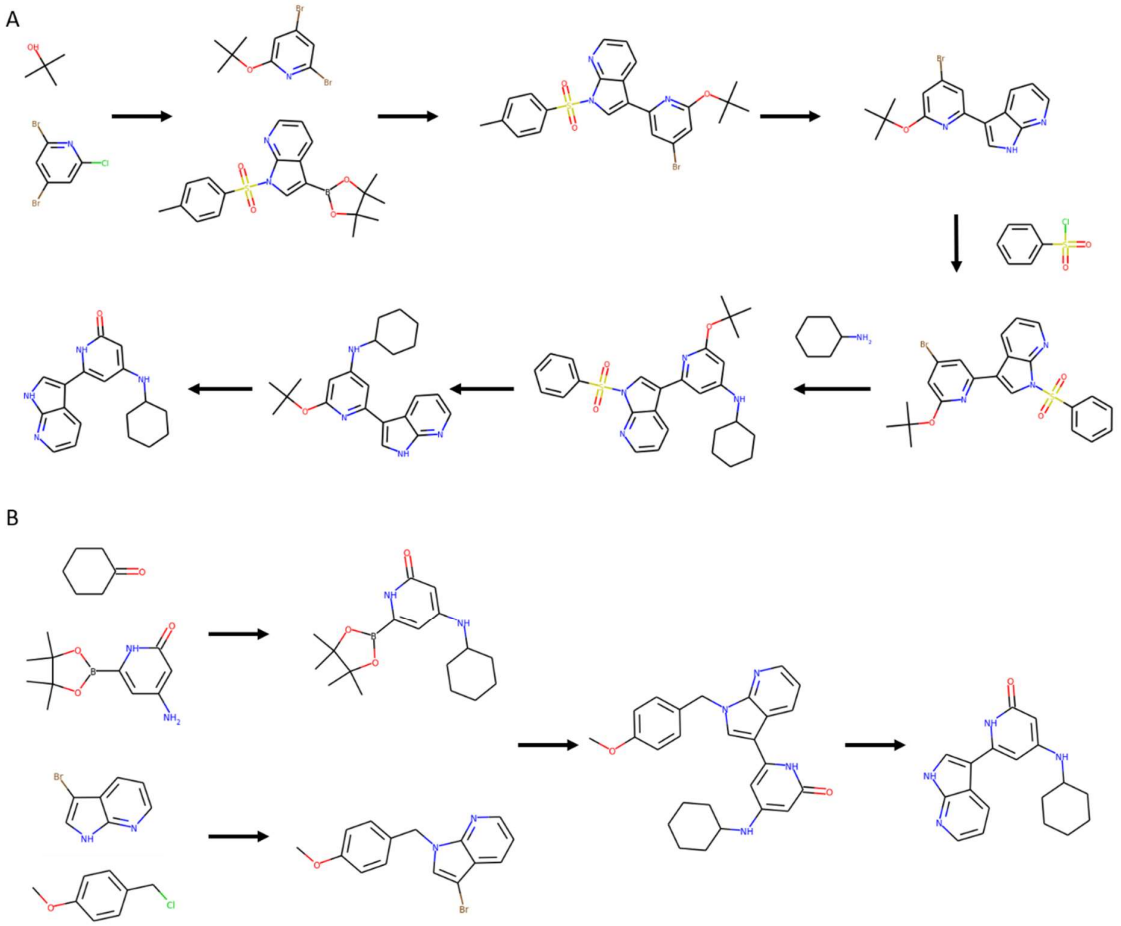


Figure 7. Example routes no 3. A) the route suggested when not using a filter, and B) the route suggested when using the *All* filter.

It is also arguable interesting to compare solved routes to partially solved (unsolved) routes, but in our experience it is very hard. For instance, the MCTS more often hits the maximum depth when it is unable to solve a route than when it finds a solution. This of course could be fixed by increasing the maximum depth, but this would increase the fraction of solved routes irrespectively if we use a filter or not. Therefore, it is hard to entangle the different search parameters into a clear causation relationship. Therefore, we did not continue with such an analysis.

Conclusions

We investigated three different ways of generating negative data and the effect this has on filter policies and route finding in a neural network guided MCTS algorithm for retrosynthetic analysis. Our proposal of using a neural network recommendation for selection of templates for negative data generation, complemented the already existing methods of using random template selection or specificity of the templates to generate negative examples. The combination of data derived with all three methods provided the most robust filter network with the best performance on all test sets. Using the filter only leads up to a small increase in the number of unsolved routes, at an acceptable increase in search time (~ 2x). Interestingly the number of nodes populated by the tree search was increased with the filters and some compounds could only be solved with the filter policies activated. Comparison of routes proposed when using a filter or when not using a filter, leads us to believe that

the filter is for the better. Although the routes suggested when using a filter might be more complex and not fully optimized, they provide a better start for further retrosynthetic planning. The filter policy trained on the combination of negative datasets will be made available as an option in the AiZynthFinder software open-sourced at https://github.com/MolecularAI/aizynthfinder

Acknowledgements

We thank Nidhal Selmi for evaluating the output of the tree search.

References

[1] Corey, E. J.; Todd Wipke, W. Computer-Assisted Design of Complex Organic Syntheses. *Science.* **1969**, *166* (3902), 178–192. https://doi.org/10.1126/science.166.3902.178.
[2] Engkvist, O.; Norrby, O.; Selmi, N.; Lam, Y.-H.; Peng, Z.; Sherer, E. C.; Amberg, W.; Erhard, T.; Smyth, L. A. Computational Prediction of Chemical Reactions: Current Status and Outlook. *Drug Discov. Today* **2018**, *23* (6). https://doi.org/10.1016/j.drudis.2018.02.014.
[3] Pensak D., A.; Corey E., J. LHASA—Logic and Heuristics Applied to Synthetic Analysis. *In Computer-Assisted Organic Synthesis, American Chemical Society*. **1977**, *61*, 1–32
[4] Ihlenfeldt, W.-D.; Gasteiger, J. Computer-Assisted Planning of Organic Syntheses: The Second Generation of Programs. *Angew. Chemie Int. Ed. English* **1996**, *34* (2324), 2613–2633. https://doi.org/10.1002/anie.199526131.
[5] Coley, C. W.; Green, W. H.; Jensen, K. F. Machine Learning in Computer-Aided Synthesis Planning. *Acc. Chem. Res.* **2018**, *51* (5), 1281–1289. https://doi.org/10.1021/acs.accounts.8b00087.
[6] Coley, C. W.; Thomas, D. A.; Lummiss, J. A. M.; Jaworski, J. N.; Breen, C. P.; Schultz, V.; Hart, T.; Fishman, J. S.; Rogers, L.; Gao, H.; Hicklin, R. W.; Plehiers, P. P.; Byington, J.; Piotti, J. S.; Green, W. H.; John Hart, A.; Jamison, T. F.; Jensen, K. F. A Robotic Platform for Flow Synthesis of Organic Compounds Informed by AI Planning. *Science (80-.).* **2019**, *365* (6453). https://doi.org/10.1126/science.aax1566.
[7] Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Hunter, C. A.; Bekas, C.; Lee, A. A. Molecular Transformer: A Model for Uncertainty-Calibrated Chemical Reaction Prediction. *ACS Cent. Sci.* **2019**, *5* (9), 1572–1583. https://doi.org/10.1021/acscentsci.9b00576.
[8] Segler, M. H. S.; Preuss, M.; Waller, P. Planning Chemical Syntheses with Deep Neural Networks and Symbolic AI. *Nature.* **2018**. *555*, 604-610 https://doi.org/10.1038/nature25978.
[9] Segler, M. H. S.; Waller, M. P. Neural-Symbolic Machine Learning for Retrosynthesis and Reaction Prediction. *Chem. - A Eur. J.* **2017**, *23* (25), 5966–5971. https://doi.org/10.1002/chem.201605499.
[10] Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Member, S.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games* **2012**, *4* (1). https://doi.org/10.1109/TCIAIG.2012.2186810.
[11] Coley, C. W.; Barzilay, R.; Jaakkola, T. S.; Green, W. H.; Jensen, K. F. Prediction of Organic Reaction Outcomes Using Machine Learning. *ACS Cent. Sci.* **2017**, *3* (5), 434–443. https://doi.org/10.1021/acscentsci.7b00064.
[12] Thakkar, A.; Kogej, T.; Reymond, J.-L.; Engkvist, O.; Bjerrum, E. J. Datasets and Their Influence on the Development of Computer Assisted Synthesis Planning Tools in the Pharmaceutical Domain. *Chem. Sci.* **2019**. https://doi.org/10.1039/C9SC04944D.
[13] Genheden, S.; Thakkar, A.; Chadimova, V.; Reymond, J.-L.; Engkvist, O.; Bjerrum, E. J. AiZynthFinder: A Fast Robust and Flexible Open-Source Software for Retrosynthetic Planning. *J. Cheminf.* **2020**, *12* https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1
[14] Coley, C. W.; Barzilay, R.; Jaakkola, T. S.; Green, W. H.; Jensen, K. F. Prediction of Organic Reaction Outcomes Using Machine Learning. *ACS Cent. Sci.* **2017**, *3* (5), 434–443.
[15] D. Lowe, Chemical reactions from US patents, 1976–Sep 2016, https://figshare.com/articles/Chemical_reactions_from_US_patents_1976-Sep2016_/ 5104873, accessed 31-04-2018.
[16] Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *J. Chem. Inf. Model.* **2010**, *50*, 742−754
[17] RDKit: Open-source cheminformatics, http://www.rdkit.org.

[18] Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[19] Schneider, N.; Lowe, D. M.; Sayle, R. A.; Landrum, G. A. Development of a Novel Fingerprint for Chemical Reactions and Its Application to Large-Scale Reaction Classification and Similarity. *J. Chem. Inf. Model.* **2015**, *55* (1), 39–53. https://doi.org/10.1021/ci5006614.

[20] Brodersen, K. H.; Ong, C. S.; Stephan, K. E.; Buhmann, J. M. The Balanced Accuracy and Its Posterior Distribution. In *Proceedings - International Conference on Pattern Recognition*; 2010; pp 3121–3124. https://doi.org/10.1109/ICPR.2010.764.

[21] Gaulton, A.; Hersey, A.; Nowotka, M.; Bento, A. P.; Chambers, J.; Mendez, D.; Mutowo, P.; Atkinson, F.; Bellis, L. J.; Cibrián-Uhalte, E.; Davies, M.; Dedman, N.; Karlsson, A.; Magariños, M. P.; Overington, J. P.; Papadatos, G., Smit, I.; & Leach, A. R. The ChEMBL database in 2017. *Nucl. acids res.*, **2017**, *45(D1)*, D945–D954. https://doi.org/10.1093/nar/gkw1074

[22] Weininger, D. SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28* (1), 31–36. https://doi.org/10.1021/ci00057a005.

[23] Coley, C. W.; Green, W. H.; Jensen, K. F. RDChiral: An RDKit Wrapper for Handling Stereochemistry in Retrosynthetic Template Extraction and Application. *J. Chem. Inf. Model.* **2019**, 59 (6), 2529–2537. https://doi.org/10.1021/acs.jcim.9b00286.

[24] Shafir, A.; Lichtor, P. A.; Buchwald, S. L. N-versus O-Arylation of Aminoalcohols: Orthogonal Selectivity in Copper-Based Catalysts. *J. Am. Chem. Soc.* **2007**, *129*, 3490–3491. https://doi.org/10.1021/ja068926f.

Supplementary information

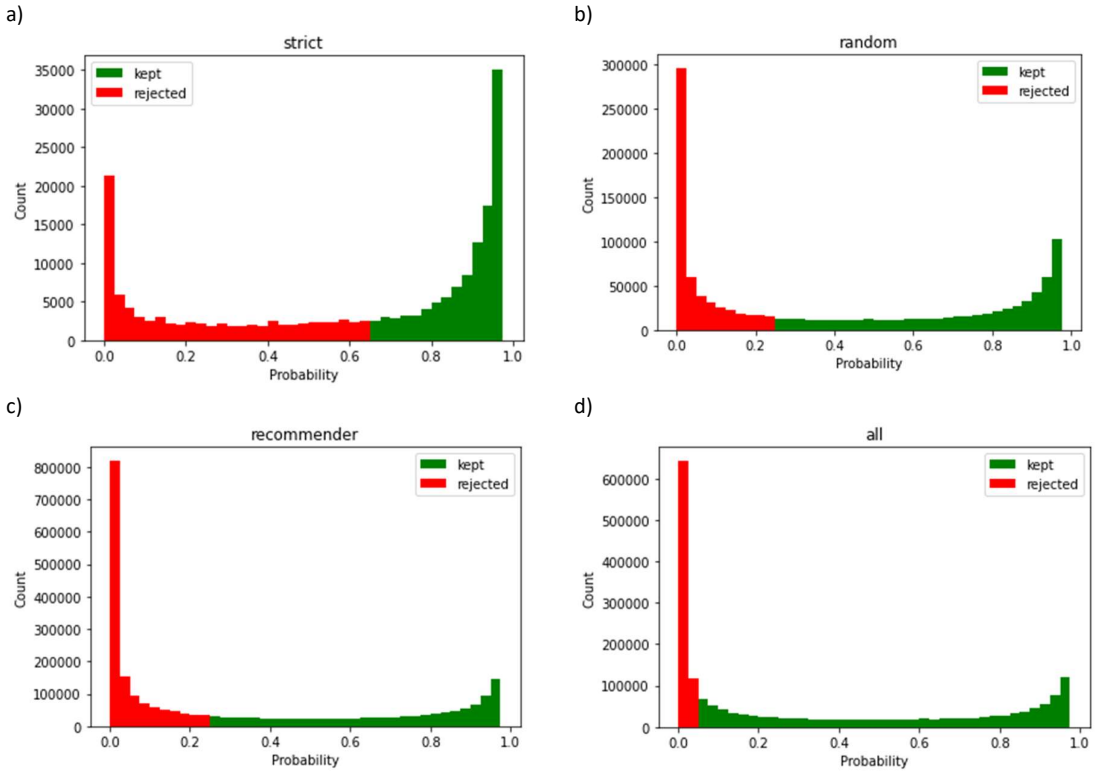
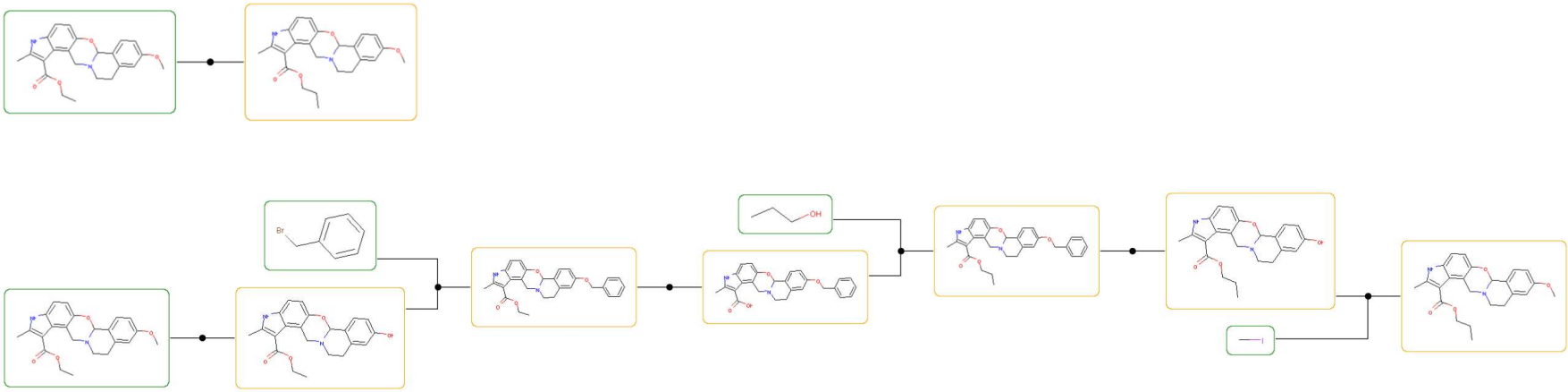


Figure S1 – the distribution of the filter probability of the reactions in the tree search.

Example of more routes

The route suggested when using no filter is shown atop the route suggested when using the *All* filter.



Here the simple transesterification reaction suggested when using no filter seems to be okay, but is rejected by the *All* filter. Instead the algorithm suggests, when the filter is applied, a long route with multiple redundant protection and deprotection steps.