Kalpa Publications in Computing

Volume 21, 2025, Pages 34–44

Proceedings of the 14th and 15th International Workshops on the Implementation of Logics

Dataset-Specific Strategies for the E Theorem Prover

Jack McKeown and Geoff Sutcliffe

University of Miami, Miami, Florida, U.S.A.
jam771@miami.edu, geoff@cs.miami.edu

Abstract

The E automated theorem proving system has an "automatic" mode that analyzes the input problem in order to choose an effective proof search strategy. A strategy includes the term/literal orderings, given clause selection heuristics, and a number of other parameters. This paper investigates the idea of creating one strategy for a given dataset of problems by merging the strategies chosen by E's automatic mode over all of the problems in the dataset. This strategy merging approach is evaluated on the MPTPTP2078, VBT, and SLH-29 datasets. Surprisingly, the merged strategies outperform E's automatic mode over all three datasets.

1 Introduction

The core component of many saturation-based Automated Theorem Proving (ATP) systems is the "given clause" algorithm [11]. This algorithm maintains two sets of clauses: a *processed* set that is initially empty, and an *unprocessed* set that initially contains the clauses from the axioms and negated conjecture. One at a time, a *given clause* is selected from the unprocessed set and brought into the processed set, then inferences are made between the given clause and other clauses in the processed set. The inferred clauses are added to the unprocessed set modulo redundancy criteria [8]. This process repeats until the empty clause is derived, the unprocessed set becomes empty, or a resource limit is reached. The derivation of the empty clause indicates that the conjecture is a theorem of the axioms, whereas an empty unprocessed set indicates that the conjecture is not a theorem of the axioms.

The saturation-based ATP system E [9] implements the DISCOUNT version [3] of the given clause algorithm. E has an automatic mode that analyzes the input problem in order to choose an effective proof search strategy. Currently, an E strategy consists of 108 key-value pairs, for the various parameters that influence the proof search. This paper investigates the idea of merging the strategies chosen by E's automatic mode for a set of problems into a single merged strategy, and using that merged strategy for all the problems.

Section 2 briefly summarizes how E performs given clause selection, how its given clause selection can be controlled by users, and how this control is defined in an E strategy. Section 3 describes how the strategy merging is performed, and describes how the merged strategies are evaluated. Section 4 describes the datasets used for evaluation, gives details about the experiments performed, and presents the experimental results. Section 5 summarizes the results, and concludes the paper.

2 Given Clause Selection in E

In E, given clause selection is guided by *clause evaluation functions (CEFs)*. Each CEF evaluates each unprocessed clause, determining a priority for each clause in a priority queue associated with the CEF. E supports a number of different CEFs, each of which is composed of an instance of a *weight function* that evaluates the clause, and a *priority function* that restricts the scope of the CEF, (Each priority function partitions the clauses in the unprocessed set so that a certain class of clauses is given preference regardless of the evaluations given by the weight function). Each weight function has a set of parameters unique to that weight function, which are provided after the priority function. Figure 1 shows an example CEF with its components labeled.

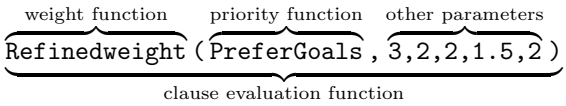

Figure 1: Example of an E clause evaluation function

During proof search CEFs are used to select the given clauses according to a *heuristic*. A heuristic is an ordered list of CEFs, each having its own integer *heuristic weight* that determines how many clauses should be selected from that CEF's priority queue before moving on to the next CEF (or back to the first CEF after the last CEF in the heuristic). Figure 2 shows an example of an E heuristic, with each line showing a CEF prefixed by its heuristic weight: the first CEF would be used once to select the given clause, then the second CEF would be used four times, then the third CEF would be used ten times, etc. This schedule would then repeat after $1 + 4 + 10 + 3 + 5 = 23$ given clause selections. The heuristic is only one part of a full E strategy. A full strategy with all 108 strategy parameters is shown in Appendix A. E strategies include a heuristic like the one shown in Figure 2 as the value of the `heuristic_def` key.

```
(1.ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1),
 4.ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5),
 10.FIFOWeight(PreferProcessed),
 3.ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1),
 5.Refinedweight(SimulateSOS,3,2,2,1.5,2))
```

Figure 2: Example of an E heuristic.

E's automatic mode for choosing a strategy is invoked using the `--auto` flag, and if invoked with the `--print-strategy` flag, E will print out the strategy in the format shown in Appendix A. Therefore, an E strategy can be saved to a file by invoking E with the `--auto` and `--print-strategy` flags and redirecting `stdout` to a file. The format of these files is similar to JSON.

3 Strategy Merging

In this work the strategies chosen by `--auto` for every problem in a given dataset are saved without attempting to solve the problems, and these saved strategies are merged in multiple ways to create other strategies that are used to solve all of the problems. This primarily means creating merged strategies that are each evaluated on all problems, but it also means

creating per-problem strategies via merging. For all 107 strategy parameters other than the `heuristic_def`, the value used in the merged strategy is the value that was used most frequently in the individual strategies. The `heuristic_def` is merged in a more sophisticated way, as follows.

This paper evaluates many ways of merging the `heuristic_def` parameter:

1. **MasterAllOnes:** The simplest way to merge the heuristics takes the union of the sets of CEFs used in all the saved strategies, and assigns a heuristic weight of 1 to each of them. This approach ignores the heuristic weights in the saved strategies as well as the number of saved strategies that each CEF occurs in.

2. **MasterWeighted:** The second way to merge the heuristics is to assign to each CEF a heuristic weight proportional to the sum of its heuristic weights from all of the saved strategies. The sums are not used directly because having very large heuristic weights would cause E to repeatedly ignore important clauses that are not preferred by a CEF being repeatedly used, but are preferred by other CEFs. Therefore the sums are scaled down by a constant factor and then rounded up using the `ceil` function. The scaling factor is determined so that the maximum heuristic weight is 20. This number was chosen as a middle ground between no scaling and aggressive scaling that would lose more information about the distribution of CEFs due to the rounding. The `ceil` function guarantees that no CEF is removed entirely from the merged heuristic.

3. **MasterWeightedRR:** To evaluate the impact of MasterWeighted's repeated use of the same CEFs, the MasterWeighted strategy is also evaluated using a modified version of E that attempts to avoid consecutively using the same CEF. It does this by going through all CEFs in a round-robin fashion. Each CEF gets a counter is initialized at its heuristic weight, and this counter is decremented when that CEF is used. When the counter reaches zero, that CEF is skipped until the heuristic resets. Once all counters reach zero, they are all reset to their heuristic weights. This is easiest to understand by example. Originally, the heuristic "3*CEF1, 2*CEF2, 1*CEF3" leads to the the following sequence of CEFs used for selection (before repeating): "1,1,1,2,2,3." Under the modified version of E, the same heuristic leads to the following sequence instead: "1,2,3,1,2,1."

4. **MasterSuccess:** Lastly, a version of the MasterWeighted strategy is created by using only the saved strategies for problems that `--auto` was able to solve. This strategy was created with the intuition that the strategies suggested by `--auto` should only be trusted to contribute to the merged strategy if they were successful on the problem for which they were suggested.

In all of the strategies, the CEFs in the merged heuristic appear in order of decreasing heuristic weight so that the "best" CEFs are first. In MasterAllOnes, where all heuristic weights are set to 1, the order is the same as in MasterWeighted.

3.1 Potential ITP Application

While the strategy merging described above could be helpful when dealing with a fixed dataset of problems, it would also be useful if a merged strategy could be evolved and applied incrementally for a growing set of related problems. This situation is encountered when ATP systems are used as "hammers" in Interactive Theorem Proving (ITP) systems [5]. A well-known example is the Isabelle [7] ITP system, whose "Sledgehammer" mode [6] submits subproblems to ATP systems like E.

5. To evaluate the potential of strategy merging for a growing set of problems, another set of strategies was created, collectively referred to as *MasterIncremental*. These strategies are created in the order that the problems are added to the set, with the kth problem getting assigned a merged strategy formed from the `--auto` strategies of the first k problems. The strategy assigned to the first problem is the same as its `--auto` strategy, and the strategy assigned to the last problem is the same as MasterWeighted. Each merging is done the same as in MasterWeighted, only with different sets of input strategies.

3.2 Ablation Study

The `heuristic_def` parameter was hypothesized to have a larger impact on the results than the other parameters in merged strategies because given clause selection is the core of the proof search. To test this hypothesis, two other methods for strategy merging were evaluated.

6. The first method, called *CommonHeuristic*, sets the `heuristic_def` parameter to its value in the MasterWeighted strategy, but keeps the value chosen by `--auto` for all the other 107 strategy parameters.

7. The second method, called *CommonElse*, is essentially the converse, keeping the value chosen by `--auto` for the `heuristic_def` parameter but setting the other 107 strategy parameters to their values in the MasterWeighted strategy.

3.3 An Auto-based Baseline

While the strategy merging can produce a strategy that generally outperforms E's automatic strategies, it is unclear whether this is due to the merged strategy being better than all of the `--auto` strategies, or if E's `--auto` mode is assigning suboptimal strategies from its set of available strategies.

8. To test this, every unique strategy that E's `--auto` mode assigns over all problems in a dataset is evaluated on all problems in the dataset. A baseline method called *AutoAll* is created by picking the best performing strategy for each problem in the dataset. Therefore, if even one strategy solves a problem, then AutoAll solves that problem. This simulates how good E's `--auto` mode could be if it perfectly picked the best strategy for each problem (from its set of available strategies).

4 Data, Experiments, and Results

All in all, nine methods for solving a set of problems were evaluated: `--auto`, AutoAll, MasterAllOnes, MasterWeighted, MasterWeightedRR, MasterSuccess, MasterIncremental, CommonHeuristic, and CommonElse. MasterAllOnes, MasterWeighted, MasterWeightedRR, and MasterSuccess each consist of a single E strategy, whereas the other methods have a different strategy for each problem.

All methods were evaluated on three datasets: MPTPTP2078, VBT, and SLH-29. The MPTPTP2078 dataset is a TPTP-compliant version of the MPTP2078 dataset [1] that consists of problems formed from the derivation of the Bolzano-Weierstrass theorem in the Mizar Mathematical Library [4]. These problems come in "bushy" and "chainy" variants, with the bushy variants having only the most immediately relevant axioms and the chainy variants having a larger set of axioms. The "bushy" variants of the problems were used in this work. The VBT

and SLH-29 datasets were both used in the CASC-J11 competition [10], and come from the Sledgehammer mode of the Isabelle theorem prover. The VBT dataset consists of 8000 problems generated by Isabelle's Sledgehammer mode from the Van Emde Boas Trees entry in the Isabelle Archive of Formal Proofs [2]. The problems are available in multiple logics, and the typed-first order versions were used here. The SLH-29 dataset is a collection of 8400 higher-order problems that also come from interactions with the Sledgehammer mode in Isabelle. For most strategy parameters the `--auto` setting is largely consistent across problems within each dataset. For example, in the MPTPTP2078, VBT, and SLH-29 datasets, only 16, 3, and 27 strategy parameters, respectively, had two or more values used in at least 5% of the problems.

The strategy merging and experimental setup are shown in Figure 3. The process is the same for all datasets:

1. E is used to save strategies for each problem using `--auto --print-strategy`.
2. The saved strategies are merged in the ways described above to get the strategies for MasterAllOnes, MasterWeighted, and MasterIncremental.
3. CommonHeuristic and CommonElse strategies are created for each problem by taking some parameter values from the saved per-problem strategies and others from the MasterWeighted strategy.
4. E is invoked on all problems using the `--auto` flag. The set of solved problems is used to select the saved strategies from step 1 that are then used to make the MasterSuccess strategy.
5. E is invoked on all problems using the MasterAllOnes, MasterWeighted, MasterSuccess, MasterIncremental, CommonHeuristic, and CommonElse methods.
6. Every strategy suggested by `--auto` is used for every problem to get the AutoAll results.

Strategies are loaded into E using the `--parse-strategy` flag, and every call to E includes the flags `--soft-cpu-limit=60` and `--cpu-limit=65`, which limit CPU time.

The results are shown in Tables 1 and 2. Table 1 shows the number of problems solved by each method. Table 2 shows the median number of given clauses selected before finding a proof, for only problems solved by all methods and excluding problems solved during presaturation interreduction (which is not guided by a strategy). In both tables the best result for each dataset is bolded. Because AutoAll is the clear winner in terms of solved problems and processed clauses for all datasets, the second best results are also bolded.

Dataset	`--auto`	Auto All	Master AllOnes	Master Weighted	Master W RR	Master Success	Master Incr.	Common Heuristic	Common Else
M'2078	1151	**1438**	**1219**	1210	1201	1199	1199	1170	1086
VBT	2637	**3596**	2701	2841	**2858**	2806	2785	2710	2521
SLH-29	3396	**4203**	3642	**3743**	3565	3505	3556	3430	3371

Table 1: Problems solved by each method

An additional perspective on the results is given by Figures 4, 5, and 6. The vertical lines in these figures show the same information as Table 1. Each row of the black and white background represents one strategy, and each column represents one problem. The rows and

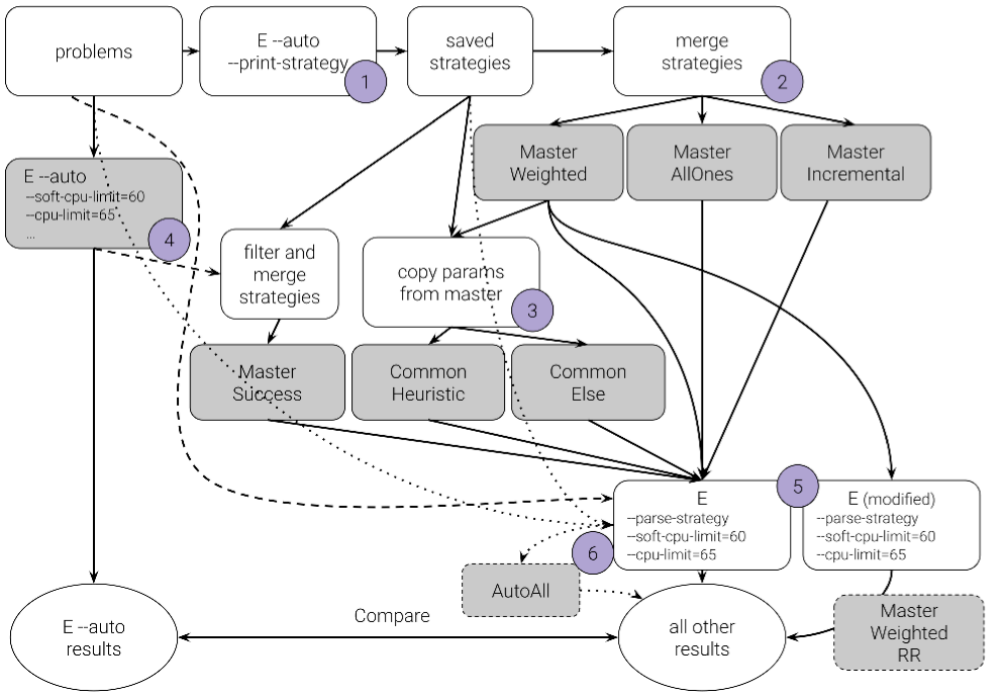

Figure 3: Strategy merging and experimental setup

Dataset	`--auto`	Auto All	Master AllOnes	Master Weighted	Master W RR	Master Success	Master Incr.	Common Heuristic	Common Else
M'2078	104	**51.5**	**92**	103.5	119.5	117	112	97	105
VBT	1794.5	**693.5**	**1475**	1565	1524	1600.5	1558	1531	1878
SLH-29	1431	**492**	874.5	**751**	827.5	760	757	932	1038

Table 2: Median number of clauses processed before finding a proof.
(For only problems solved by all methods)

columns are sorted such that the strategy that solves the most problems appears on top and the problems that are solved by the most strategies appear on the left. For instance, the fact that the transition from black to white occurs earlier in Figure 5 than in Figure 4 suggests that the problems in the VBT dataset are harder on average than the problems in the MPTPTP2078 dataset.

The MasterAllOnes and MasterWeighted strategies both improve upon `--auto` in terms of number of problems solved and processed clauses on all datasets. The MasterWeighted strategy solves more problems than the MasterAllOnes strategy on the VBT and SLH-29 datasets, but not on the MPTPTP2078 dataset. Perhaps this is because the MPTPTP2078 problems are

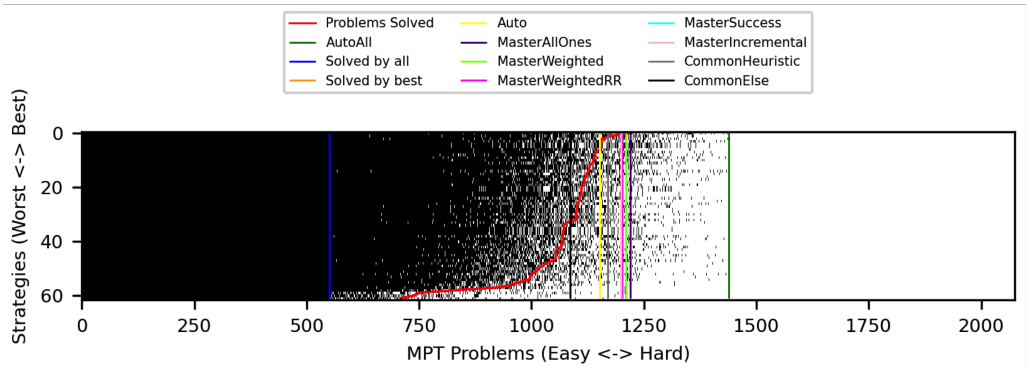

Figure 4: MPTPTP2078 Experiment Results

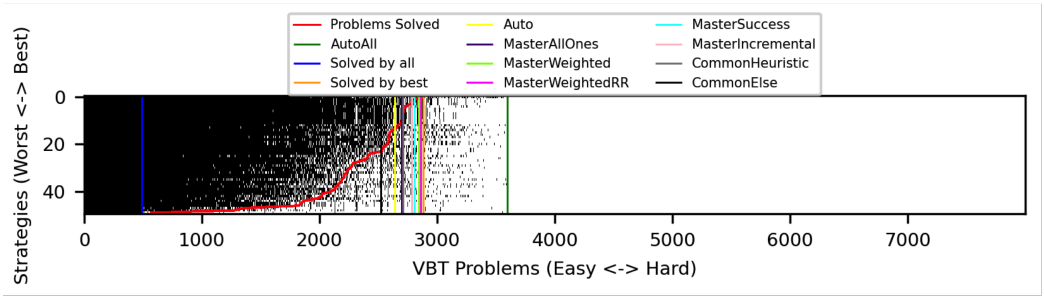

Figure 5: VBT Experiment Results

solved in fewer given clause selections on average. The MasterAllOnes strategy uses more unique CEFs in the short-term, but the MasterWeighted strategy ostensibly uses the CEFs in better proportions in the long-term. Even if many CEFs agree that an important clause should be selected, the repeated use of a single CEF in the MasterWeighted strategy could delay the clause's selection. In such a case, the MasterAllOnes strategy would select the important clause more quickly. As problem difficulty increases, however, this potential delay would represent a smaller proportion of the total selections needed to find a proof. This was the motivation behind the MasterWeightedRR method. The results were mixed, however, with MasterWeightedRR solving fewer problems than both MasterWeighted and MasterAllOnes on the MPTPTP2078 and SLH datasets, but more than both the VBT dataset.

The AutoAll results suggest that E's auto mode could be improved by selecting a more effective strategy for each problem without modifying the underlying set of candidate strategies that E's `--auto` mode uses. Additionally, the AutoAll result provides context that merged strategies are not universally better than the individual strategies, although they are better on average than the particular ones chosen by E's `--auto` mode.

The MasterSuccess strategies perform worse than both the MasterWeighted strategy and MasterAllOnes strategy in terms of both number of problems solved and processed clauses. This is surprising because this strategy is constructed by merging only the strategies that were successful in solving their associated problem. Perhaps failure to solve a problem is more of

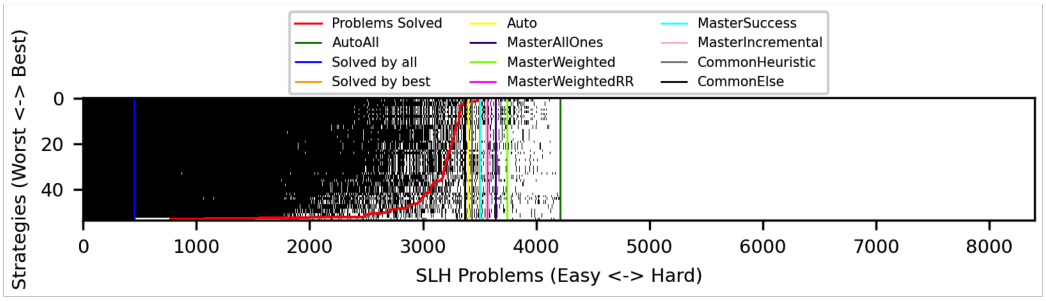

Figure 6: SLH Experiment Results

an indication that the problem is difficult than it is an indication that the strategy chosen by `--auto` is bad.

The MasterIncremental strategies perform worse than MasterWeighted and MasterAllOnes on all datasets in terms of both number of problems solved and processed clauses. In light of the general success of strategy merging, this was unsurprising because fewer strategies are being merged to create each MasterIncremental strategy than were merged in MasterAllOnes or MasterWeighted. That being said, MasterIncremental solves more problems than `--auto` on all three datasets and uses fewer processed clauses (median) on the VBT and SLH-29 datasets, suggesting that incremental strategy merging could be useful within ITP "hammers".

The CommonHeuristic strategies outperform the `--auto` strategies on each dataset, whereas the CommonElse strategies do not, except for in terms of the number of clauses processed on the SLH-29 dataset. This suggests a coupling between the 107 merged non-`heuristic_def` strategy parameters and the merged `heuristic_def` parameter. The merged non-`heuristic_def` parameter values are beneficial, but only when used in conjunction with the merged `heuristic_def` parameter. (It cannot be the case that the merged heuristic is the only helpful merged parameter, because MasterWeighted outperforms CommonHeuristic.)

5 Conclusion

This paper demonstrates that, at least for the three datasets used here, it is possible to improve upon E's automatic strategy by merging the strategies that E automatically chooses, and then using the merged strategy for all of the problems. While this approach would likely be less effective over a very diverse dataset, this strategy merging seems to be a helpful way to inject helpful bias for a homogenous dataset. Additionally, incremental strategy merging shows promise for incorporation into ITP tools like Sledgehammer.

References

[1] J. Alama, D. Kühlwein, E. Tsivtsivadze, J. Urban, and T. Heskes. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *CoRR*, abs/1108.3446, 2011.

[2] T. Ammer and P. Lammich. van Emde Boas Trees. *Archive of Formal Proofs*, November 2021. https://isa-afp.org/entries/Van_Emde_Boas_Trees.html, Formal proof development.

[3] Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. DISCOUNT: A System for Distributed Equational Deduction. In *Rewriting Techniques and Applications: 6th International Conference, RTA-95 Kaiserslautern, Germany, April 5–7, 1995 Proceedings 6*, pages 397–402. Springer, 1995.

[4] G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, K. Pak, and J. Urban. Mizar: State-of-the-art and Beyond. In *Intelligent Computer Mathematics - International Conference, CICM July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015.

[5] J. Blanchette, C. Kaliszyk, L. Paulson, and J. Urban. Hammering Towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.

[6] Jia Meng and Lawrence C. Paulson. Translating Higher-Order Clauses to First-Order Clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

[7] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283. Springer Science & Business Media, 2002.

[8] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[9] S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, Higher, Stronger: E 2.3. In *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[10] G. Sutcliffe and M. Desharnais. The 11th IJCAR Automated Theorem Proving System Competition - CASC-J11. *AI Commun.*, 36(2):73–91, 2023.

[11] A' Voronkov. Algorithms, Datastructures, and Other Issues in Efficient Automated Deduction. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 13–28. Springer-Verlag, 2001.

A Example Strategy

Here is the strategy chosen by E for the MPT0001+1.p problem from the MPTPTP2078 dataset, given as an example of a strategy file. The whitespace around the heuristic has been adjusted for readability, so it might not work within E without edits.

```
{
  {
     ordertype:               KBO6
     to_weight_gen:           precedence
     to_prec_gen:             invfreqhack
     rewrite_strong_rhs_inst: true
     to_pre_prec:             ""
     conj_only_mod:           0
     conj_axiom_mod:          0
     axiom_only_mod:          0
     skolem_mod:              0
     defpred_mod:             0
     force_kbo_var_weight:    false
     to_pre_weights:          ""
     to_const_weight:         0
     to_defs_min:             false
     lit_cmp:                 1
     lam_w:                   20
     db_w:                    10
```

```
   ho_order_kind:           lfho
}
no_preproc:                      false
eqdef_maxclauses:                20000
eqdef_incrlimit:                 20
formula_def_limit:               24
sine:                            "Auto"
add_goal_defs_pos:              false
add_goal_defs_neg:              false
add_goal_defs_subterms:         false
heuristic_name: Default
heuristic_def: "(
  1.ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1),
  4.ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5),
  1.FIFOWeight(PreferProcessed),
  1.ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1),
  4.Refinedweight(SimulateSOS,3,2,2,1.5,2)
)"
prefer_initial_clauses:          false
selection_strategy:              SelectComplexExceptUniqMaxHorn
pos_lit_sel_min:                 0
pos_lit_sel_max:                 9223372036854775807
neg_lit_sel_min:                 0
neg_lit_sel_max:                 9223372036854775807
all_lit_sel_min:                 0
all_lit_sel_max:                 9223372036854775807
weight_sel_min:                  0
select_on_proc_only:             false
inherit_paramod_lit:             false
inherit_goal_pm_lit:             false
inherit_conj_pm_lit:             false
enable_eq_factoring:             true
enable_neg_unit_paramod:         true
enable_given_forward_simpl:      true
pm_type:                         ParamodSim
ac_handling:                     1
ac_res_aggressive:               true
forward_context_sr:              true
forward_context_sr_aggressive:   false
backward_context_sr:             false
forward_subsumption_aggressive: false
forward_demod:                   2
prefer_general:                  false
condensing:                      false
condensing_aggressive:           false
er_varlit_destructive:           true
er_strong_destructive:           true
er_aggressive:                   true
split_clauses:                   0
split_method:                    0
split_aggressive:                false
split_fresh_defs:                true
rw_bw_index_type:                FP7
```

```
   pm_from_index_type:             FP7
   pm_into_index_type:             FP7
   sat_check_grounding:            ConjMinMinFreq
   sat_check_step_limit:           5000
   sat_check_size_limit:           9223372036854775807
   sat_check_ttinsert_limit:       9223372036854775807
   sat_check_normconst:            false
   sat_check_normalize:            false
   sat_check_decision_limit:       10000
   filter_orphans_limit:           9223372036854775807
   forward_contract_limit:         9223372036854775807
   delete_bad_limit:               2000000000
   mem_limit:                      0
   watchlist_simplify:             true
   watchlist_is_static:            false
   use_tptp_sos:                   false
   presat_interreduction:          true
   detsort_bw_rw:                  false
   detsort_tmpset:                 false
   arg_cong:                       all
   neg_ext:                        off
   pos_ext:                        off
   ext_rules_max_depth:            -1
   inverse_recognition:            false
   replace_inj_defs:               false
   lift_lambdas:                   true
   lambda_to_forall:               true
   unroll_only_formulas:           true
   elim_leibniz_max_depth:         -1
   prim_enum_mode:                 pragmatic
   prim_enum_max_depth:            -1
   inst_choice_max_depth:          -1
   local_rw:                       false
   prune_args:                     false
   preinstantiate_induction:       false
   fool_unroll:                    true
   func_proj_limit:                0
   imit_limit:                     0
   ident_limit:                    0
   elim_limit:                     0
   unif_mode:                      single
   pattern_oracle:                 true
   fixpoint_oracle:                true
   max_unifiers:                   4
   max_unif_steps:                 256
}
```