Online supplement for: A Study of Quality and Accuracy Tradeoffs in Process Mining, by Zan Huang and Akhil Kumar (Appendices A –F)

APPENDIX A

This example illustrates the calculation of the mismerge score

Type	[C, D] = 1	[D, C] = 1	[C, D] = 0	[D, C] = 0	[C, s] = [D, s] = [s, C] = [s, D] = 1	[C, s]= [D, s] [s, C]= [s, D]	Mismerge score
'P'	No	No	-	-	-	No for s = A, L2	4
'C'	-	-	Yes	Yes	-	No for s = A, L2	2
'S'	No	-	-	Yes	No for [A, D], [L2, D], [Start, C], [Start, D], [End, C], [End, D] ([C, s] = 1 → s = A, B, L2, End; [s, D] = 1 → s = Start, B, L1)	-	7
'L'	No	No	-	-	No for [A, D], [L2, D], [Start, C], [Start, D], [End, C], [End, D] ([D, s] = 1 → s = A, B, L2, End; [s, D] = 1 → s = Start, B, L1)	-	8

Table A1: Calculation of mismerge score for possible merges between tasks C and D

Table A1 shows the calculation for the mismerge score using the heuristic rules for the possible merges of tasks C and D. Each column represents a condition of a rule and a row shows how a condition applies to a type of merge relationship (S, C, S, or P). A '–' means that a condition does not apply. A 'No' entry is marked for each condition violation, and the 'No' entries are added up to get the mismerge score that reflects the unsuitability of a candidate merge with respect to the AdjM matrix of the log. The four proposed merges in Table A1 for tasks C and D will result in four incomplete models (with one block and four tasks) as level-2 nodes in the search tree, each with a mismerge score equal to the mismerge score of the merge resulting in the incomplete model. For each level-2 node, a new AdjM matrix is constructed by merging the rows and columns corresponding to C and D in the original AdjM matrix. Then the same procedure of proposing and evaluating possible merges can be applied on level-*k* nodes to create level-*k*+1 nodes in the search tree until level *N*+1 where complete process models are constructed.

Appendix B

This is a long version of the **Extract_Model** Algorithm for finding the model with the highest quality among a given number of most probable candidate models with desired accuracy from a log.

Algorithm Extract_Model (Log, Taskset, **Global** Max_Limit, **Global** Desired_Accuracy)
Build AdjM; set **Global** model_count = 0; set **Global** candidates = {};
model0.tuples = {}; model0.AdjM = AdjM; model0.MMscore = 0; evaluated_models = {}
//a tuple is (task/block i, task/block j, relationship R, mismerge (MM) score)
SearchModel(model0, Taskset, Log)
Function SearchModel(model, Taskset, Log, evaluated_models)
If (model_count > Max_limit) then Return model* in evaluated_models with highest Q; **Exit**
tupleList = **ProposePair**(model.AdjM)
remove model from candidates
Foreach (tuple **in** tupleList)
model1.tuples = model.tuples ∪ tuple
model1.MMscore = model1.MMscore + tuple.MMscore
model1.AdjM **= Merge**(AdjM, model2.lastTuple.i, model2.lastTuple.j)
candidates = candidates ∪ model1
sort candidates on MMScore
model2 = candidates.pop(0)
If (all Tasks are included in model2)
SLOP_set = **Replay**(Log, Model)
Remove (100-AP)% elements from SLOP_set \\ simple heuristic is currently employed to choose the elements that contain uncommon SL/OP structures to remove
SLOP = SLOP[1] ∪ SLOP[2] ∪ ….
model2.Q = Quality(model2, SLOP)
evaluated_models = evaluated_models ∪ model2
Else: **SearchModel**(model2, Taskset, Log)
End SearchModel
Function ProposePair(AdjM)
newList = {}
Foreach possible merge (i, j, R) from AdjM:
If (R == 'P') MMScore = (# of mismatches in AdjM between the rows and columns for i, j) +
(1 –AdjM[i][j]) + (1 – AdjM[j][i])
If (R == 'C') MMScore = (# of mismatches in AdjM between the rows and columns for i, j) +
AdjM[i][j] +AdjM[j][i]
If (R == 'S') MMScore = # of zeros in {AdjM[i][j], AdjM[i][s], AdjM[j][s], AdjM[s][i], AdjM[s][j]}
(where s ≠ i, j and (AdjM[i][s] == 1 or AdjM[s][j] == 1))
If (R == 'L') MMScore = # of zeros in {AdjM[i][j], AdjM[j][i], AdjM[i][s], AdjM[j][s], AdjM[s][i],
AdjM[s][j]} (where s ≠ i, j and (AdjM[s][j] == 1 or AdjM[j][s] == 1))
newList = newList ∪ (i, j, R, MMScore) //this is a list of tuples
Return(List)
End ProposePair
Function Merge(AdjM, i, j)
AdjM = AdjM[][] ∪ {**OR**(row i, row j)} ∪ {**OR**(col i, col j)} – {row i, row j, col. i, col j}
Return AdjM
End Merge

Function Replay(model, Log, Accuracy)
Compute *next_tasks*0 as a list of tasks that are possible to be the first to execute; SLOP_set = {}
Foreach log instance *l*:
  occurred = {}; *next_tasks* = *next_tasks*0; SLOP_l = {}
  Foreach entry *e* in *l*:
   If (*e* **in** *next_tasks*)
      update *next_tasks* to be possible tasks to see next; *occurred* = *occurred* ∪ {*e*};
   Else
       If (*e* **in** *occurred*)
            Find the appropriate block *b* in model to set as self-loop to make *e* reachable;
            Add *b_s*l into SLOP_l
     Else
            Find the appropriate block *b* in model to set as optional such that the updated *next_tasks* contains *e*
       Add *b_op* into SLOP_l; update *next_tasks* accordingly
  Add SLOP_l to SLOP_set
Return SLOP_set
End Replay

APPENDIX C

Example 1: To illustrate how our algorithm works, we ran it on the log of Table 1 in the paper. This example is in two parts. In the first part, we ran our algorithm on the first 9 log instances of Table 1. Table C1(a) shows the first adjacency matrix made from this log.

Table C1. Initial AdjM matrix and revised AdjM matrix after merging tasks C and D

(a) AdjM matrix

	Start	A	B	C	D	L1	L2	End
Start	0	1	1	1	1	0	0	0
A	0	0	0	1	0	1	0	1
B	0	0	0	1	1	1	0	1
C	0	1	1	0	0	0	1	1
D	0	1	1	0	0	0	1	1
L1	0	0	1	1	1	0	1	0
L2	0	0	1	1	0	1	0	0
End	0	0	0	0	0	0	0	0

(b) Merged AdjM

	S	A	B	C-D	L1	L2	End
St	0	1	1	1	0	0	0
A	0	0	0	1	1	0	1
B	0	0	0	1	1	0	1
C-D	0	1	1	0	0	1	1
L1	0	0	1	1	0	1	0
L2	0	0	1	1	1	0	0
End	0	0	0	0	0	0	0

Our algorithm merged C and D in the first step. The merged matrix is shown in Table C2. After subsequent merges it produced the following model shown in Figure C1 (same as Figure 3 in the paper).

Model.tuples =
{(A, B, 'Choice', Block1),
(C, D, 'Choice', Block2),
(Block1, L1, 'Loop', Block3),
(Block2, L2, 'Loop', Block4),
(Block3, Block4, 'Parallel', Process)}

Model.SLOP = {B_sl, Block2_op}

(a) A process tree (b) Tuples and vectors

Figure C1: Model representation as tree and as merge tuples and SL/OP markings

The above model was based on the first 9 instances of Table C1 and does not contain any SL or OP tasks. Next we ran our algorithm to include all 12 instances of Table C1 because the last three instances require use of SL and OP structures. In this case, we were able to find the correct model (exactly as in Figure 1) at the 431st candidate model that was considered. Table C2 shows how the **Replay** function operates while scanning each log for conformance with the model. The last row of this table shows a composite bit vector combining the three vectors through bitwise

OR operation. From this row it is clear that task B is in a self-loop and block B1 (including tasks C and D) is optional. So, this is the final model shown in Figure 1 in the paper.

Table C2: Self-loop and optional markings created by **Replay** function for log of Table 1

Log instance	**SLOP_1**
A	Block2_op
D A L1 B B	B_sl
A L1 A	Block2_op
Union	B_sl, Block2_op

APPENDIX D

We first create a simulated process model, and then use it to generate a log. We start with a list of tasks and then pick two tasks at random to be combined in one of four basic structures (sequence, choice, parallel and loop). As an example, in Figure D1 tasks t3 and t6 are chosen from 10 tasks to combine in a P structure to create block b1, and then two other blocks b2 and b3 are formed similarly. Next b2 and b3 are combined to form b4, while t1 and b1 combine to form b5. In this way, pairs of unattached tasks or blocks are selected randomly and further combined until there is one large block that represents the final process. We then also randomly mark certain tasks and blocks as SL and OP structures. When generating logs from the simulated model, for a choice structure a branch is chosen with equal probability (0.5); for a loop structure 1-3 times of iterations are executed; for a parallel structure the tasks under the two parallel branches are mixed randomly into a single sequence.

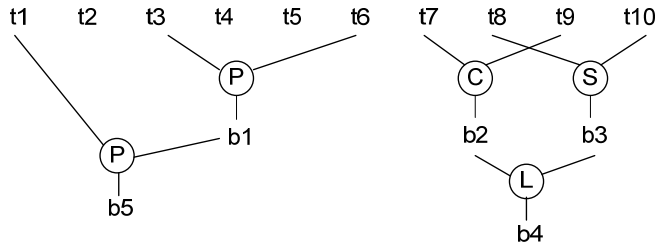


Figure D1: Generating a process model

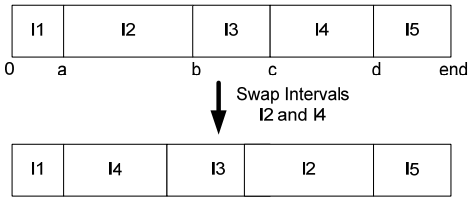


Figure D2: Simulating noise

In addition to the above experiments on noise-free logs, we also conducted experiments on simulated logs with *artificially induced noise*. We define a general noise operation on a log instance as (see Figure D2):

$$row = row[0{:}a] + row\,[c{:}d] + row\,[b{:}c] + row\,[a{:}b] + row[d{:}end]$$

where a, b, c, d: are randomly chosen positions of log entries in the log such that $a \leq b \leq c \leq d$. As an example, consider a log entry or trace like "t1 t7 t9 t8 t6 t4 t3 t4 t2". If a = 1, b = 2, c=5, d=7, then after performing this operation, the log entry would be "t1 t4 t3 t4 t8 t6 t7 t9 t2" which is an example of noisy log trace. This is a general transformation resulting in a swap of any pair of segments from a log instance. In our experiments, we randomly chose 5% of the instances or rows in the simulated logs and performed the noise operation up to three times on them. Correspondingly, we set the target accuracy to be 95% when running the process mining algorithm on these noisy logs. However, a target accuracy of 100% was used with the noise-free logs.

APPENDIX E

Table E1 shows the top-10 most frequent tasks in our data. When counting the frequency, we considered the multiple occurrences within the same log instance. The task names are self-explanatory for the most part. Notice of allowance is a notification to the applicant that they are entitled to a patent under the law and requesting payment of a specified issue fee. It should also be noted that these were all instances in which patents were eventually issued, in some cases after first being rejected. However, these rejections were "non-final".

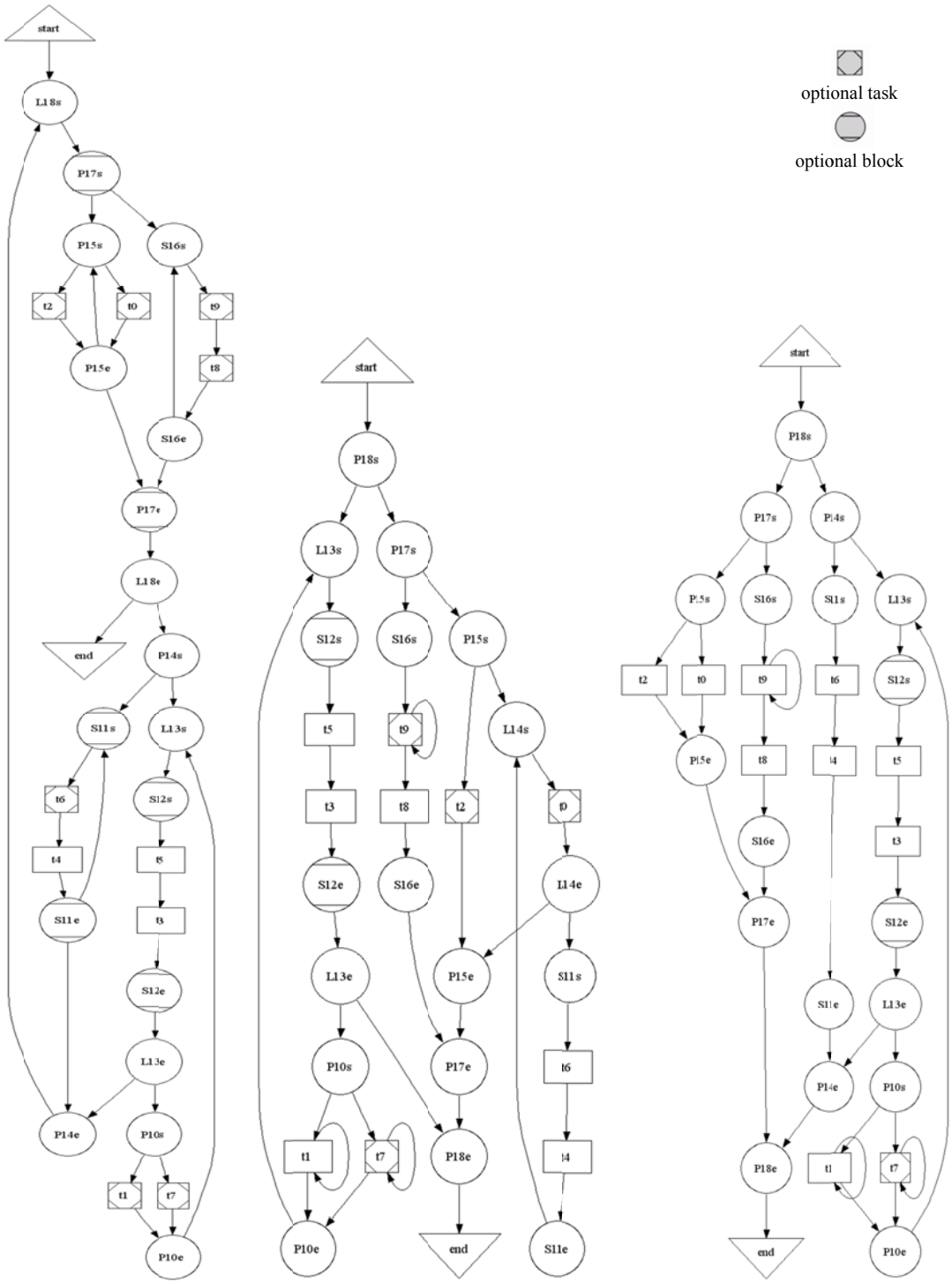
Figure E1 shows three process models for Experiments 1, 2, and 8 of Table 4. The labels "S", "L", "C", and "P" denote the corresponding types of blocks. Each split node (with suffix 's') has a corresponding join or end node (with suffix 'e'). Thus, P17s is a parallel split node which matches with P17e. A legend is provided alongside to indicate the symbols used for OP tasks and blocks. The self-loop tasks and blocks are indicated using a self-loop in these diagrams. Across the three models, we observe consistent local structures such as the sequence structure consisting of t6 and t4, the sequence structure consisting of t5 and t3, the parallel structure consisting of t1 and t7. Moving up to a higher level, we see that the parallel block consisting of t0, t2, t8, and t9 and the loop block consisting of t5, t3, t1 and t7 were present across the four models as well. While these local structures make intuitive sense for non-experts, many differences are also there between the models which reflect the presence of noise.

Table E1: Top 10 Most Frequent Tasks

Task Id	Task Description	Frequency
0	IFW (Image File wrapper) Scan & PACR (Patent Application Capture and Review) Auto Security Review	31,922
1	Issue Fee Payment Verified	33,608
2	Issue Notification Mailed	31,976
3	Mail Non-Final Rejection	42,067
4	Mail Notice of Allowance	32,968
5	"Non-Final" Rejection	42,067
6	Notice of Allowance Data Verification Completed	32,967
7	Receipt into Publications	76,931
8	Recordation of Patent Grant Mailed	31,799
9	Response after Non-Final Action	42,281

The three models vary in how the local structures are assembled into the complete model. Model (a) differs from the other two models in that the root-level block in this model is a loop

structure with an OP parallel sub-block. There are also many OP and SL structures at the lower level blocks/tasks clearly suggesting it is a low quality model. Model (b) has a parallel structure at the root consisting of the loop sub-block with t5, t3, t1 and t7, and the parallel sub-block comprising of the remaining tasks. The sequence structure consisting of t6 and t4 is now within a loop structure together with t0. The resulting model successfully explained 97.55% log instances with 9 SL structures and 10 OP structures, and a quality metric of 0.6528. Model (c) differs from Model (b) in that the non-OP sequence block consisting of t6 and t4 becomes one of four parallel branches from the root.



(a) (b) (c)

Figure E1: Patent application process models: (a) accuracy = 100%; Q = 0.4974 (b) accuracy = 97.55%; Q = 0.6528 (c) accuracy = 68.63%; Q = 0.6558.

APPENDIX F

We have made our algorithm implementation available online together with the patent log data we used in this paper: http://www.personal.psu.edu/faculty/h/u/huz2/pm/HKMining.htm