Approximate Achievability in Event Databases

Austin Parker*, Gerardo I. Simari**, Amy Sliva, and V.S. Subrahmanian

Department of Computer Science and UMIACS
University of Maryland College Park, College Park, MD 20742, USA

Abstract. An event DB is a database about states (of the world) and events (taken by an agent) whose effects are not well understood. Event DBs are omnipresent in the social sciences and may include diverse scenarios from political events and the state of a country to education-related actions and their effects on a school system. We consider the following problem: given an event DB $\mathcal{K}$ representing historical events (what was the state and what actions were done at various past time points), and given a goal we wish to accomplish, what "change attempts" can the agent make so as to "optimize" the potential achievement of the goal? We define a formal version of this problem and derive results on its complexity. We then present a basic algorithm that provably provides a correct solution to finding an optimal state change attempt, as well as an enhanced algorithm that is built on top of the well known trie data structure and is also provably correct. We show correctness and algorithmic complexity results for both algorithms and report on experiments comparing their performance on synthetic data.

1 Introduction

A large number of well known data sets in the social sciences have a tabular form. Each row refers to a period of time, and each column represents a variable that characterizes the state of some entity during a time period. These variables naturally divide into those actionable variables we can control (which we will call "action variables") and those we cannot (which we will call "state variables"). For example, data sets regarding school performance for various U.S. states contain "state variables" such as the graduation rate of students in the state and the student to staff ratio during some time frame, while the "action" variables might refer to the level of funding provided per student during that time frame, the faculty salary levels during that time period, etc. Clearly, a U.S. state can attempt to change the levels of funding per student and/or change the faculty salaries in an attempt to increase the graduation rate. In a completely different setting, political science data sets about the stability of a country (such as the data sets created by the well known Political Instability Task Force [2]) may have "state variables" such as the Gross Domestic Product (GDP) of a country during a time period, the infant mortality rate during the same time period and the number of people killed in political conflict in the country during that time period, while "action" variables might include information about the investment in hospitals or education during that time frame, the number of social workers available, and so forth. A government might want to see what actionable

* Current affiliation: Institute for Defense Analysis Center for Computer Science, MD, USA.
** Current affiliation: Computing Laboratory, Oxford University, United Kingdom.

W. Liu (Ed.): ECSQARU 2011, LNAI 6717, pp. 737–748, 2011.

policies it can attempt to achieve a certain goal (*e.g.*, bringing the infant mortality rate below some threshold).

These are just two examples of problems that are not easily solved using current algorithms for reasoning about actions in AI or by AI planning systems. The main reasons are the following (i) the relationships between the actions and their impact on the state are poorly understood, (ii) a set of actions, taken together, might have a cumulative effect on a state that might somehow be more than a naive combination of the effects of those actions individually—which of course are not known anyway, and (iii) the actions under consideration may not succeed—an attempt to raise hospital funding may be blocked for reasons outside of anyone's control.

In this paper, we first propose (Section 2) the notion of an *event DB* (this is not novel, but generalizes several social science data sets). Section 3 defines the concept of "state change attempts" (SCAs for short) and formulates the problem of finding "optimal" SCAs towards a given goal; we present results on the computational complexity of finding optimal SCAs. In Section 4, we first present a straightforward algorithm called DSEE_OSCA to compute optimal SCAs, and then develop a vastly improved algorithm called TOSCA based on tries in Section 5. Though tries are a well known data structure, the novelty of our work is rooted in how TOSCA uses tries to solve optimal SCA computation problems with lower computational complexity. Finally, in Section 6, we briefly describe an implementation of both algorithms, together with an experimental analysis to demonstrate that TOSCA is quite tractable on data sets of reasonable size.

2 Preliminaries on Event DBs

An event DB is a relational database whose rows correspond to some time period (explicit or implicit) and whose columns are of two types—*state attributes* and *action attributes*. Throughout this paper, we assume the existence of some arbitrary, but fixed set $\mathbf{A} = \{A_1, \ldots, A_n\}$ of action attributes, and another arbitrary, but fixed set $\mathbf{S} = \{S_1, \ldots, S_m\}$ of state attributes. As usual, each attribute (state or action) A has a domain $dom(A)$, which in this work we assume to be finite. A *tuple* w.r.t. $(\mathbf{A}, \mathbf{S})$ is any member of $dom(A_1) \times \cdots \times dom(A_n) \times dom(S_1) \times \cdots \times dom(S_m)$. We use $t(S_i)$ (resp. $t(A_j)$) in the usual way to denote the value assigned to attribute S_i (resp. A_j) by a tuple. An *event database* $\mathcal{K}$ is a finite set of tuples w.r.t. $(\mathbf{A}, \mathbf{S})$. We assume all attributes A have domain $dom(A) \subset \mathbb{R}$. We use $\mathcal{A}$ to represent the set $dom(A_1) \times \cdots \times dom(A_n)$ and $\mathcal{S}$ to represent the set $dom(S_1) \times \cdots \times dom(S_m)$. We say a tuple is an *action tuple* if it contains only values for the action attributes and that it is a *state tuple* if it contains only values for the state attributes.

Example 1. Throughout this paper, the "School" data set is a data set from the U.S. State Education Data Center about U.S. school performance. Figure 1 presents a small part of this data set; we call it the *school* event DB. The columns labeled A_1, ..., A_4 represent action attributes, while the columns labeled S_1, ..., S_5 represent state attributes.

The school dataset contains nine attributes explained at the bottom of Figure 1. Math and reading scores obtained from standardized tests are combined into one annual *proficiency score*. School administrators's have the goal of increasing proficiency and graduation percentages by certain amounts.

	A_1	A_2	A_3	A_4	S_1	S_2	S_3	S_4	S_5
t_1:	9,532	61.6	7.8	4.2	81.1	49.1	51.3	50.6	Yes
t_2:	9,691	63.2	7.8	5.7	82.3	52.1	54.6	53.3	No
t_3:	9,924	63.8	8.1	3.1	82.0	59.8	60.4	60.1	Yes
t_4:	10,148	64.2	7.6	3.4	83.4	60.5	64.2	63.3	Yes
t_5:	10,022	64.0	7.2	2.9	83.2	63.9	68.9	66.9	Yes

Fig. 1. Small instance of an event *DB* containing hypothetical school performance data. Action variables are A_1: Funding ($/Student), A_2: Salaries (% of Total Funding), A_3: Student/Staff Ratio, A_4: Proficiency Increase Target; state variables are S_1: Graduation (%), S_2: Math Proficiency, S_3: Reading Proficiency, S_4: Proficiency Score, S_5: Target Reached.

3 Optimal State Change Attempts

In this section, we formalize the notion of a state change attempt (SCA). The idea is that when an SCA is successfully applied to a given tuple, it changes the action attributes with the hope of these changes resulting in a change in the state. For example, decreasing class size may lead to better proficiency scores.

Definition 1. *A* simple SCA *is a triple* (A_i, vf, vt) *where* $vf, vt \in Dom(A_i)$ *for some* $A_i \in \mathbf{A}$. *A (non-simple)* SCA *is a set* $\{(A_{i_1}, vf_1, vt_1), \ldots, (A_{i_k}, vf_k, vt_k)\}$ *of simple SCAs such that* $i_j \neq i_k$ *for all* $j \neq k$.

When clear from context, we will refer to these concepts as *simple changes* and *changes*, respectively. Intuitively, a *simple* SCA modifies one attribute, while a state change attempt may modify more than one.

Definition 2. *Given a tuple* t*, an action attribute* A_i*, and* $vf, vt \in Dom(A_i)$*, a simple SCA* (A_i, vf, vt) *is* applicable *w.r.t.* t *iff* $t(A_i) = vf$. *The* result of applying a simple SCA *that is applicable w.r.t.* t *is a tuple* t' *where* $t'(A_i) = vt$ *and* $t'(A_j) = t(A_j)$ *for all attributes (action and state)* $A_j \neq A_i$. *We use* $\gamma(t, (A_i, vf, vt))$ *to denote tuple* t'.

A state change attempt $SCA = \{(A_{i_1}, vf_1, vt_1), \ldots, (A_{i_k}, vf_k, vt_k)\}$ *is* applicable *w.r.t.* t *iff all* (A_{i_j}, vf_j, vt_j) *for* $1 \leq j \leq k$ *are applicable to w.r.t.* t.

The application of SCA to t will be denoted with $\gamma(t, SCA)$; note that an SCA only changes action attributes.

Example 2. A simple SCA w.r.t. the school data from Example 1 could be the following: $a_1 = (A_1, 8700, 8850)$, *i.e*, funding is increased from $8,700 to $8,850 per student, or $a_2 = (A_2, 62.3, 65)$, *i.e.*, salaries are increased from 62.3% to 65% of the budget. Let $SCA = \{a_1, a_2\}$ be an SCA. If we assume that the values of the action attributes in the current environment are $t = (8700, 64, 7, 3.2)$, then a_1 is applicable w.r.t. t, but a_2 is not. The result of applying a_1 to t is $\gamma(t, (A_1, 8700, 8850)) = t' = (8850, 64, 7, 3.2)$.

The result of applying an SCA is therefore the result of applying each simple change. However, these changes do not occur without cost.

Definition 3. *Let* $a = (A_i, vf, vt)$ *be a simple state change attempt. The* cost of attempting a *is given by a real-valued function cost* : $\{A_1, \ldots, A_m\} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, *where* $cost(A_i, vf, vt)$ *is the cost of changing action attribute* A_i *from* vf *to* vt.

Cost functions will be highly dependent on the application domain, and we assume them to be provided by a user. The *cost* of an attempt, $cost(SCA) = \sum_{a \in SCA} cost(a)$, is the sum of the costs of the simple state change attempts in SCA.

Example 3. Let $a_1 = (A_1, 8700, 8850)$, $a_2 = (A_2, 62.3, 65)$ be the same simple changes from Example 2, and $a_3 = (A_4, 3.8, 3.9)$ be a third simple change (*i.e.*, increment the proficiency increase target from 3.8 to 3.9). A possible cost function could be defined in terms of monetary cost, in which: $cost(a_1) = 150 * s$ (where s is a constant set to the number of students affected), $cost(a_2) = 2.7 * A_1$, and $cost(a_3) = 0$.

Thus far, we have studied SCAs that are always successful. However, in general, we cannot expect this to be the case—the funding per student may not change simply because one attempted to change it. We will assume state change attempts are only probabilistically successful—they only induce the change attempted according to a specified probability. Further, we will assume that the probability of any simple change occurring successfully depends on the entire set of changes attempted.

Example 4. Consider the situation described in Example 3. Here the state change attempt a_2 increases teacher salaries from 62.3% to 65%. On its own, attempting this change may anger taxpayers (who would pay for the increase) and may only have a 10% probability of succeeding. Likewise, increasing per student funding might have a 15% chance of success. However, if the taxpayers are willing to increase teacher salaries, then they may also tend to approve per student funding increases, perhaps leading to a joint probability of 9% that both of these will occur when attempted together.

Let SCA and $SCA' \subseteq SCA$ be SCAs; let $pOccur(SCA'|SCA)$ denote the probability that only the actions in SCA' occur given that SCA is attempted. Such probabilities can either be derived from historical data or be explicitly stated by a user. When we say that a state change attempt SCA is "attempted" for a tuple t describing the current situation, this means that each $SCA' \subseteq SCA$ has the chance $pOccur(SCA'|SCA)$ of being successful, *i.e.*, of having $\gamma(t, SCA')$ be the resulting tuple.

Effect Estimators. The goal of this paper is to allow an end user to take an event DB $\mathcal{K}$ and a goal G (some desired outcome condition on state attributes) and find an SCA that "optimally" achieves goal G. We assume w.l.o.g. that goals are expressed as standard conjunctive selection conditions [8] on state attributes. We now define *effect estimators*.

Definition 4. *For action tuple t and goal G, an* effect estimator *is a function $\varepsilon(t, G) \rightarrow [0, 1]$ that maps a tuple and a goal to a probability $p \in [0, 1]$.*

Intuitively, $\varepsilon(t, G)$ specifies the conditional probability of goal G holding given that we are in a situation where the action attributes are as specified in t. This quantity can be estimated in many ways, some of which will be investigated below.

Probabilistic State Change Effectiveness. As mentioned above, we assume an environment where, just because an SCA is performed, it is not necessarily the case that all parts of the SCA will actually accomplish the attempted change. When one attempts to change the situation via SCA, any subset of SCA may succeed. For instance, if one tries to decrease the student/staff ratio and increase the funding per student, perhaps the

student/staff ratio increases as expected, but that the funding per student remains the same. Thus to truly gauge the effectiveness of a state change attempt, we must consider the probability of each subset of the attempt occurring.[1]

Definition 5. *The probability of a state change attempt* $SCA = \{ (A_{i_1}, vf_1, vt_1), \ldots, (A_{i_k}, vf_k, vt_k) \}$ *satisfying goal* G *when applied to action tuple* t *is pEff*$(t, G, SCA, \varepsilon) = \sum_{SCA' \in \mathcal{P}(SCA)} pOccur(SCA'|SCA) \cdot \varepsilon(\gamma(t, SCA'), G)$*, where* $\mathcal{P}(SCA)$ *denotes the power set of* SCA*.*

pEff(t, G, SCA, ε) is computed by summing over all the state changes that may occur given the attempt of SCA: since any subset can occur, this summation ranges over $SCA' \subseteq SCA$. For each SCA' that may occur, one multiplies its probability of occurring given that SCA was attempted $\big(pOccur(SCA'|SCA)\big)$ times the effectiveness of the given attempt according to ε (recall that $\gamma(t, SCA')$ is the action tuple resulting from the application of SCA' to the original action tuple t). The following result shows that for arbitrary effect estimators, computing state change effectiveness is intractable.

Proposition 1. *For condition* G*, state change attempt* SCA*, action tuple* t*, and effect estimator* ε*, deciding if pEff*$(t, G, SCA, \varepsilon) > 0$ *is NP-hard w.r.t.* $|\mathbf{A}|$*. If* $\varepsilon(.)$ *can be computed in PTIME w.r.t.* $|\mathbf{A}|$*, the problem is NP-complete.*[2]

The Highest Probability SCA (HPSCA) Problem. Let $\mathbf{A} = \langle A_1, \ldots, A_n \rangle$ and $\mathbf{S} = \langle S_1, \ldots, S_m \rangle$, $\mathcal{K}$ be an event DB, t be an action tuple describing the current values of the action attributes, G be a goal over $\mathbf{S}$, *cost* and *pOccur* be the functions as mentioned earlier, and $p \in [0, 1]$ be a real number; does there exist a change attempt SCA such that *pEff*$(t, G, SCA, \varepsilon) \geq p$?

The above problem is stated as a decision problem; a search problem, to *find* such an SCA, can be analogously stated. We refer to any state change attempt that is a solution to this problem as an *optimal state change attempt* (OSCA, for short).

Theorem 1. *If the effect estimator used can be computed in PTIME, the Highest Probability SCA problem is* $\#P$*-hard and in PSPACE w.r.t.* $|\mathbf{A}|$*.*

The $\#P$-hard reduction uses $\#SAT$ (the language $\{\langle F, n \rangle\}$, where F is a formula with exactly n solutions), and membership in PSPACE is shown by giving algorithms.

A Basic Algorithm. We will now provide a basic algorithm to solve the HPSCA problem. It works by first enumerating each possible state change attempt with size at most h, then choosing the one that has the highest probability. Since there are only $O(|\mathbf{A}|^h)$ such state change attempts, this algorithm runs in $O(|\mathbf{A}|^h)$, which is PTIME with respect to the number of action attributes $|\mathbf{A}|$.

Proposition 2. *Algorithm 1 runs in time in* $O(|\mathbf{A}|^h)$*, and returns* (SCA, c, ef) *where* $|SCA| \leq h$*,* $c = cost(SCA)$ *and* $ef =$ *pEff*(t, G, SCA, ε)*, such that there is no other* (SCA, c', ef') *with* $c' < c$ *and* $ef' > ef$*.*

To extend this technique to the non-limited, general version of the problem, one simply needs to solve the limited version of the problem with h equal to $|\mathbf{A}|$:

[1] In this work, we assume that each simple change attempt either succeeds or fails completely.

[2] NP-hardness is shown via reduction from subset sum.

Algorithm 1: solveHPSCA$(t, G, \varepsilon, h, p)$
1. Let $R = \emptyset$ // the set to be returned.
2. Add $(\emptyset, 0, pEff(t, G, \emptyset, \varepsilon))$ to R. // Initialize R with empty state change attempt.
3. For each $A_i \in \mathbf{A}$
4. For each value $v \in dom(A_i)$
5. **continue** if $v = t(A_i)$ // Go to next value, t won't be changed by this SCA.
6. // iterate over all members of R, growing those which are small enough.
7. For each $(SCA, c, ef) \in R$
8. **continue** if $|SCA| = h$.
9. Let $SCA' = SCA \cup \{(A_i, t(A_i), v)\}$.
10. Let c' be the cost of SCA' and ef' be $pEff(t, G, SCA', \varepsilon)$.
11. Add (SCA', c', ef') to R.
12. **return** $(SCA, c, ef) \in R$ s.t. $ef \geq p$ and $\nexists (SCA, c', ef') \in R$ with $c' < c$ and $ef' > ef$; false otherwise.

Fig. 2. Returns (SCA, c, ef), where c is the cost of state change attempt SCA and ef is the probability of effectiveness of SCA s.t. ef is highest and c is lowest

4 Different Kinds of Effect Estimators

In this section we introduce several effect estimators which specify the likelihood of a given action tuple satisfying a given goal condition G. An effect estimate answers the question: "if I succeed in changing the environment in this way, what is the probability that this new environment satisfies my goal?"

Learning Algorithms as Effect Estimators. We now show how to take any supervised learning algorithm (*e.g.*, neural nets, decision trees, etc.) and apply it to the event database $\mathcal{K}$ to get an effect estimator. We abstractly model a machine learning algorithm as a *learner*, which, given the appropriate information, will produce a *classifier*.

Definition 6. *For event DB $\mathcal{K}$ and goal condition G, a* classification algorithm *is a function* learner $: (\mathcal{K}, G) \mapsto$ classifier*, where* classifer *is a function from action tuples to the interval* $[0, 1]$. *Given a classification algorithm learner, a* learned effect estimator *is defined to be* $\varepsilon_{lrn}(learner, \mathcal{K})(t, G)$*, returning* $learner(\mathcal{K}, G)(t)$.

For instance, neural networks [7] fit this definition: we first define learner to be a function that generates a neural network with input nodes for each action attribute and exactly one output node with a domain of $[0, 1]$. The learner function then trains the network via backpropagation according to $\mathcal{K}$ and G. The resulting network is the classifier function, and will, given a set of values for the action attributes, return a value in the interval $[0, 1]$. We can use a classification algorithm to create a *learned effect estimator*.

Data Selection Effect Estimators. In this section we examine the special case of an effect estimator that uses selection operations in a database to create an estimation. For our purposes, selection operations will be denoted $\sigma_G(\mathcal{K})$, where $\mathcal{K}$ is an event DB and G is some goal condition on the state tuples. $\sigma_G(\mathcal{K})$ returns the subset of $\mathcal{K}$ satisfying the condition G.

Definition 7. *For goal G and action tuple t, a data selection effect estimator is a function that takes an event DB $\mathcal{K}$ as input and returns an effect estimator: $\varepsilon^* : \mathcal{K} \mapsto$*

$(t, G) \mapsto p$, *where* $p \in [0, 1]$. *We require that* ε^* *be implemented with a fixed number of selection operations on* $\mathcal{K}$ *and that* $\varepsilon^*(\mathcal{K})(t, G)$ *be 0 if there is no tuple in* $\mathcal{K}$ *whose action attributes match* t.

A data selection effect estimator differs from a normal effect estimator in that it depends explicitly on selection from event DB $\mathcal{K}$. While data selection effect estimators are limited to using only selection operators we will see that there are many ways to specify the relationship between G and the situation described by t using only selection operations. We abuse the notation used for selection operators in databases by writing $\sigma_t(\mathcal{K})$ to denote the selection of all the tuples in $\mathcal{K}$ that have the values described by t for the corresponding attributes.

Definition 8. *The* data ratio effect estimator *is defined:* $\varepsilon_r^*(\mathcal{K})(t, G) \stackrel{def}{=} \frac{|\sigma_{t \wedge G}(\mathcal{K})|}{|\sigma_t(\mathcal{K})|}$ *whenever* $|\sigma_t(\mathcal{K})| > 0$, *and zero otherwise.*

The data ratio effect estimator returns the marginal probability of G occurring given that the values specified by the action tuple t occur.

Example 5. Suppose we have a school metrics database containing only three columns: class size, teacher salary and graduation rate. The class size and teacher salary are action attributes, while the graduation rate is a state attribute. We want to determine from the data what fraction of the time a graduation rate is at least 95% for an average class size of 20 and an average teacher salary of \$60,000. According to ε_r^*, this fraction is the fraction of tuples in the database with class size 20 and teacher salary \$60,000 that have a graduation rate over 95% divided by the total number of tuples in the database with class size 20 and teacher salary \$60,000.

One important feature of the data ratio effect estimator is that when there is no information on a given tuple, it assumes the tuple to be a negative instance. This allows it to quickly eliminate possibilities not contained in the database, and reduces the search space needed to compute optimal state change attempts. Further examples of data selection effect estimators include cautious or optimistic ratio effect estimators, which take the confidence interval into account.

Definition 9. *The cautious ratio effect estimator returns the probability of* G *given* t *to be the low end of the 95% confidence interval:* $\varepsilon_{c95}^*\mathcal{K}(t, G) \stackrel{def}{=} \varepsilon_r^*(\mathcal{K})(t, G) - 1.96 \cdot \sqrt{\frac{\varepsilon_r^*(\mathcal{K})(t,G)(1-\varepsilon_r^*(\mathcal{K})(t,G))}{|\sigma_t(\mathcal{K})|}}$ *(if* $\sigma_t(\mathcal{K})$ *is empty,* $\varepsilon_{c95}^*(\mathcal{K})(t, G)$ *is defined to be zero).*

There is a whole class of cautious ratio effect estimators: one for every confidence level (90%, 80%, 99%, etc.). There are also optimistic ratio effect estimators which return the high end rather than the low end of the confidence interval.

Since data selection effect estimators are computed via a finite number of selection operations, effect estimators can always be computed in time in $O(|\mathcal{K}|)$. The complexity of finding SCAs changes when we insist on using data selection effect estimators. Problems that were NP-complete or $\#P$-hard w.r.t. the size of the action schema are polynomial in $|\mathcal{K}|$ when only data selection effect estimators are allowed.

Algorithm 2: DSEE_OSCA(DB $\mathcal{K}$, Goal G, Action tuple *env*, p)
1. Let $Dat1 = \emptyset$ // $Dat1$ will contain state change attempts and their probability of occurrence.
2. // Iterate through all tuples satisfying G in $\mathcal{K}$.
3. For $t \in \sigma_G(\mathcal{K})$ do // Create SCA s.t. $\gamma(env, SCA)$ equals t on action attributes.
4. $SCA = \{(A, env(A), t(A)) \mid env(A) \neq t(A)\}$
5. If $(SCA, \cdot) \in Dat1$ then continue. // Already visited
6. Let $f = \varepsilon_r^*(\mathcal{K})(t, G)$.
7. Add (SCA, f) to $Dat1$.
8. Let $Dat2 = \emptyset$
9. For $(SCA, f) \in Dat1$ do
10. Let $nextF = pOccur(SCA|SCA) \cdot f$.
11. For $(SCA', f') \in Dat1$ do
12. If $SCA' \subsetneq SCA$ then
13. $nextF = nextF + pOccur(SCA'|SCA) \cdot f'$
14. Add (SCA, ef) where $(SCA, nextF)$ to $Dat2$.
15. Remove any (SCA, ef) from $Dat2$ where $ef < p$.
16. **return** $\arg\min_{(SCA,ef) \in Dat2}(cost(SCA))$.

Fig. 3. A brute force algorithm for solving the HPSCA problem

Proposition 3. *For goal G, state change attempt SCA, action tuple t, and event DB $\mathcal{K}$, if the effect estimator ε^* is a data selection effect estimator then deciding whether pEff$(t, G, SCA, \varepsilon^*(\mathcal{K})) > 0$ takes $O(|\mathcal{K}|^2)$ time.*

Theorem 2. *If the effect estimator is a data selection effect estimator, then the HPSCA problem can be solved in $O(|\mathcal{K}|^2)$ time.*

Computing OSCAs with Data Selection Effect Estimators. Using data selection effect estimators, we can devise algorithms to find optimal SCAs. In this section we use only the data ratio effect estimator (Definition 8). Figure 2 presents the DSEE_OSCA algorithm to solve the HPSCA problem.

Proposition 4. *Algorithm 2 computes SCA such that pEff$(env, G, SCA, \varepsilon_r^*(\mathcal{K})) \geq p$ and there is no other feasible state change attempt SCA' such that $cost(SCA') < cost(SCA)$ and pEff$(env, G, SCA', \varepsilon_r^*(\mathcal{K})) \geq p$.*

The DSEE_OSCA algorithm works by selecting all tuples in the event DB $\mathcal{K}$ satisfying the goal condition, then adding the pair (SCA, f) to a data structure $Dat1$ where f is the chance that SCA, when successful, results in a state satisfying the goal G (*i.e.*, $\varepsilon_r^*(\mathcal{K})(t, G)$). In the next loop, two things happen: (i) f is multiplied by the probability that SCA is successful, and (ii) we iterate through all state change attempts and sum the probability of occurrence of each subset of SCA with that subset's probability of satisfying the goal G, adding the result to data structure $Dat2$. At this point $Dat2$ contains pairs (SCA, ef), where ef is the probability of effectiveness of SCA according to Definition 5. The algorithm then prunes all state change attempts without sufficiently high probabilities of effectiveness, and returns the one with the lowest cost.

Proposition 5. *Algorithm 2 runs in time $O(|\mathcal{K}|^2)$.*

Alg. 3: TOSCA(Trie T, Goal G, Action tuple env, p)

1. Let $Dat1$ = TOSCA-Helper(T, G, env).
2. Let $Dat2 = \emptyset$.
3. For $(SCA, f) \in Dat1$ do
4. Let $nextF = pOccur(SCA|SCA) \cdot f$.
5. For $(SCA', f) \in Dat1$ do
6. If $SCA' \subsetneq SCA$ then
7. $nextF = nextF + pOccur(SCA'|SCA) \cdot f'$
8. Add (SCA, ef) with $(SCA, nextF)$ to $Dat2$.
9. Remove any (SCA, ef) from $Dat2$ where $ef < p$.
10. **return** $\arg\min_{(SCA,ef)\in Dat2}(cost(SCA))$.

Fig. 4. Computes a state change attempt with minimal cost and probability of effectiveness at least p using a trie.

Alg. 4: TOSCA-Helper(Trie T, Goal G, Action tuple env)

1. If T is a leaf node // Similar to Algorithm 2...
2. Let $Dat = \emptyset$
3. For $t \in \sigma_G(tuples(T))$
4. // Create SCA s.t. $\gamma(env, SCA) = t$
5. $SCA = \{(A, env(A), t(A))|t(A) \neq env(A)\}$
6. If $(SCA, \cdot) \in Dat$ then continue to next t
7. $f = \varepsilon^*_r(tuples(T))(t, G)$.
8. Add (SCA, f) to Dat.
9. **return** Dat.
10. Else // Recursively call for all children of T.
11. Let $(A, Edges) = T$.
12. **return** the set

$$\cup_{(v^-, v^+, N)\in Edges} \textit{TOSCA-Helper}(N, G, env)$$

Fig. 5. Returns a set of (SCA, v) pairs, where SCA is a state change attempt and v is $\varepsilon^*(\mathcal{K})(G, Sit = \gamma(SCA, env))$.

5 Trie-enhanced Optimal State Change Attempt (TOSCA)

In this section, we present the TOSCA algorithm that uses tries [3] to improve the performance of finding an optimal state change attempt. In TOSCA, a trie is used to index the event DB to reduce the search space necessary for the data selection effect estimator in the DSEE_OSCA algorithm (Figure 3). An internal trie node is a pair $(Atr, Edges)$ where $Atr \in \mathbf{A} \cup \mathbf{S}$ is an attribute and $Edges$ contains (v^-, v^+, N) pairs, where v^- and v^+ are values from $Dom(Atr)$ with $v^- < v^+$ and N is another trie node. A leaf node in a trie maintained by TOSCA is simply a set of tuples from the DB, denoted $tuples(N)$. Tries have a unique root node.

A trie is *data correct* if for any leaf node N there is a unique path from the root $(Atr_1, Edges_1), \ldots, (Atr_{k-1}, Edges_{k-1}), N$ such that for all $t \in tuples(N)$ and all i between 1 and $k-1$, there is $(v^-, v^+, (Atr_{i+1}, Edges_{i+1})) \in Edges_i$ such that $v^- \leq t(Atr_i) < v^+$. That is, the path to a leaf node determines which tuples are stored there. A trie is *construction correct* if for all sibling nodes (v_1^-, v_1^+, N_1) and (v_2^-, v_2^+, N_2), $v_1^- \geq v_2^+$ or $v_2^- \geq v_1^+$.

The *Trie-enhanced Optimal State Change Attempt* (TOSCA) algorithm uses tries to reduce the average case run time for computing optimal state change attempts. TOSCA is divided into the *base* and a *helper*, Algorithms 3 and 4 (Figures 4 and 5) respectively.

Example 6. In our example run of Algorithm 3, we use a simple database containing four tuples $\{(A_1 = 1, S_1 = 1), (A_1 = 2, S_1 = 1), (A_1 = 3, S_1 = 0), (A_1 = 3, S_1 = 1)\}$, and the trie T pictured in Figure 6. We use the tuple $(A_1 = 0)$ as the action tuple *env*, the goal condition $S_1 = 1$, and the threshold 0.7 as p. The first step of Algorithm 3 is to create $Dat1$ via Algorithm 4, which recursively traverses the trie, beginning at node A. At node B, Algorithm 4 recognizes a leaf node and selects tuples from that node that satisfy the goal condition, iterating through them in turn beginning with $(A_1 = 1, S_1 = 1)$. The state change attempt that changes the environment tuple

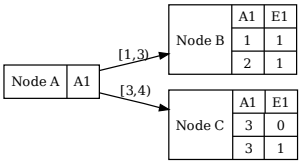

Fig. 6. The trie used in Example 6

$(A_1 = 0)$ to $(A_1 = 1, S_1 = 1)$ is $SCA = \{(A_1, 0, 1)\}$. The time saving step of the algorithm now occurs at line 7, where we run ε_r^* on the database $tuples(T)$ instead of the entire database (line 6 of Algorithm 2). Because there is only one tuple in $tuples(T)$ with $A_1 = 1$, and because that tuple also satisfies the goal condition, f is set to 1 and $(\{(A_1, 0, 1)\}, 1)$ is added to Dat. Similarly, $(\{A_1, 0, 2\}, 1)$ is added on the next tuple: $(A_1 = 2, S_1 = 1)$, finishing the call to node B.

The call to node C has slightly different results. The only member of $tuples(T)$ to satisfy the goal condition is $(A_1 = 3, S_1 = 1)$. Further, ε_r^* produces a result of $1/2$, as of the two tuples with value 3 for A_1, only one of them satisfies the condition that $S_1 = 1$. The returned set from this recursive call contains only $(\{(A_1, 0, 3)\}, 1/2)$.

After merging all recursive calls, the set $\{\ (\{(A_1, 0, 3)\}, 1/2),\ (\{(A_1, 0, 2)\}, 1),\ (\{(A_1, 0, 1)\}, 1)\ \}$ is returned and labeled $Dat1$ by Algorithm 3. The next loop multiplies the second value of each member of $Dat1$ by the probability of the associated state change attempt occurring, which is provided by a user *a priori* and we will assume to be $3/4$ for all state change attempts. The inner loop then adds the probabilities associated with subsets of the state change attempt (of which there are none in this example). This results in the data structure $Dat2$ consisting of pairs $(SCA, pEff(env, S_1 = 1, SCA, \varepsilon_r^*))$, or $\{(\{(A_1, 0, 3)\}, 3/8), (\{(A_1, 0, 2)\}, 3/4), (\{(A_1, 0, 1)\}, 3/4)\}$.

At this point, those members of $Dat2$ with too low a probability of effectiveness are eliminated (only $(\{A_1, 0, 3\}, 3/8)$) and the SCA with lowest cost is returned.

Proposition 6. *Algorithm 3 computes SCA s.t. pEff*$(env, G, SCA, \varepsilon_r^*(\mathcal{K})) \geq p$ *and there is no other feasible state change attempt* SCA' *such that* $cost(SCA') < cost(SCA)$ *and pEff*$(env, G, SCA', \varepsilon_r^*(\mathcal{K})) \geq p$.

The worse case time complexity of Algorithm 3 is $O(|\mathcal{K}|^2)$. However, the complexity of Algorithm 4 is $O(|\mathcal{K}| \cdot k)$, where k is the size of the largest leaf node in trie T. Since Algorithm 4 replaces the loop on line 9 of Algorithm 2 —a loop that takes time $O(|\mathcal{K}|^2)$ —we can expect speedup proportional to $k/|\mathcal{K}|$. Since, in the average case, k will be $|\mathcal{K}|/2^h$, (h is the trie's height) this speedup can be large.

While k is bounded by $|\mathcal{K}|$, it is usually much smaller: on the order of $|\mathcal{K}|/2^h$ for a trie of height h. We expect $Dat1$ to have size $O(|\mathcal{K}|)$, as it will be the same as $Dat1$ on line 9 of Algorithm 2. It was produced by at most $2 \cdot |\mathcal{K}|/k$ recursive calls to Algorithm 4 (there are at most $2 \cdot |\mathcal{K}|/k$ nodes in the trie). When given a leaf node, Algorithm 4 takes time in $O(k^2)$. Thus the run time of Algorithm 4 is in $O(|\mathcal{K}| \cdot k)$. The loop on line 3 then runs in time in $O(|\mathcal{K}|^2)$ (it is the same loop as in Algorithm 2), resulting in an overall run time in $O(|\mathcal{K}|^2)$. However, we will see that in practice, substantial speedup is achieved by using the $O(|\mathcal{K}| \cdot k)$ Algorithm 4 rather than $O(|\mathcal{K}|^2)$.

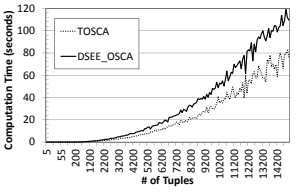

Fig. 7. Average running times for DSEE_OSCA and TOSCA over synthetic data, varying # of tuples

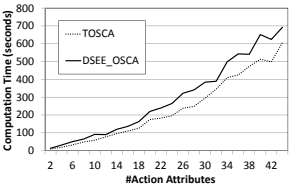

Fig. 8. Varying # of action attributes; Fixed: # of tuples at $8,000$

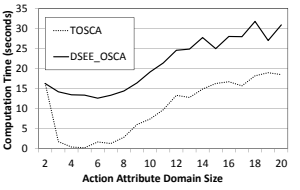

Fig. 9. Varying action attb. dom. size; Fixed: # tuples at $8K$, # sit. attb. at 4, and # event attb. at 3

6 Experimental Evaluation

We performed experiments to answer two main questions, with the following setup. We automatically generated k tuples with 4 action attributes and 3 state attributes. Each tuple's value for the action attributes was chosen randomly from $[0, 1]$. To generate the values for the state attribute, we generated random boolean formulas over the action attributes consisting of the operators $<, >, =, \neq$, and $\wedge$. We allowed at most three "$\wedge$" connectives in each formula. In a given tuple, each state attribute value is set to 1 if its associated formula is satisfied by the action attributes in that tuple, and set to 0 otherwise. Because we have the formula defining the state attributes, we can check the accuracy of the state change attempts returned by each algorithm. To do this, we apply the state change attempt and determine the state attribute values. The accuracy of a given algorithm will be the fraction of the time the resulting values for the state attributes satisfy the goal condition. Unfortunately, due to space limitations, we cannot include here experiments evaluating accuracy; we will provide a more comprehensive experimental analysis, including results on real world data, in future work.

Question 1: Which techniques scale best w.r.t number of tuples? We want to how DSEE_OSCA and TOSCA scale when presented with large amounts of data, *i.e.*, number of tuples. In these experiments, we provided the algorithms with 1,000 to 10,000 tuples. The results in Figure 7 show TOSCA to perform better than DSEE_OSCA as the database increases in size. Note that TOSCA does have a pre-computation step whose running time has been left out of these figures. However, the time needed to compute the trie is several orders of magnitude smaller than the running time of TOSCA, with only 91 ms to construct a trie with 10K tuples.

Question 2: Which techniques scale best w.r.t. number of attributes and their domain size? Figure 8 shows how DSEE_OSCA and TOSCA scale as the number of attributes increases in a database with $8,000$ tuples. This graph shows TOSCA outperforming DSEE_OSCA; it is important because the trie in TOSCA should lose efficiency as the number of attributes increases (the trie's depth equals the number of attributes). However, this shows that the decrease in the trie's efficiency does not affect the ability of the trie to offer TOSCA a speedup. Finally, Figure 9 shows TOSCA also outperforming DSEE_OSCA when the domain size of action attributes is varied.

7 Related Work and Conclusions

There is substantial work in the AI-planning community on discovering sequences of actions that lead to a given outcome (sometimes specified as a goal condition similar to this work), see [5] for an overview. However, AI planning assumes the effects of actions to be explicitly specified. Similarly, another related area is that of Reasoning about Actions [1,6]; work in this area generally assumes that descriptions of effects of actions on fluent predicates, causal relationships between such fluents, and conditions that enable actions to be performed are available. Our work approaches a similar problem in a fundamentally different and data-driven way, assuming *(i)* actions only change certain parameters in the system, *(ii)* all attempted changes succeed probabilistically depending on the set of attempted changes, and *(iii)* the effects of the changed parameters on the state can only be determined by appeal to past data. Finally, research within the Machine Learning community on the problem of classification [4] is also related to our endeavor. The main differences between that research and our own is that we are not only interested in classifying situations in past data (this is actually aided by the fact that goal conditions are provided), but in how to *arrive once again at similar situations*. As we have seen, this also involves analyzing costs of performing actions and their probabilities of success.

In this paper we have shown that determining optimal state change attempts is not an easy problem, since we prove that the optimization task belong to complexity classes widely believed to be intractable. However, we show that TOSCA is provably correct, and report preliminary experimental results on synthetic data showing that it is faster than a basic solution and tractable for reasonably sized inputs.

In future work, we will provide a more comprehensive empirical evaluation, including results on real world data and accuracy; finally, we will also investigate other interesting variants of the problem of finding optimal state change attempts.

Acknowledgements. The authors were funded in part by AFOSR grant FA95500610405 and ARO grant W911NF0910206. This work was also partially supported by the European Research Council under the EU's 7th Framework Programme (FP7/2007-2013)/ERC grant 246858 – DIADEM.

References

1. Baral, C., Tuan, L.-c.: Reasoning about actions in a probabilistic setting. In: AAAI 2002, pp. 507–512. AAAI Press, Menlo Park (2002)
2. Davies, J.L., Gurr, T.R.: Preventive Measures: Building Risk Assessment and Crisis Early Warning Systems. Rowman and Littlefield (1998)
3. Fredkin, E.: Trie memory. Communications of the ACM 3(9), 490–499 (1960)
4. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
5. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann, San Francisco (2004)
6. Pearl, J.: Reasoning with cause and effect. AI Mag. 23(1), 95–111 (2002)
7. Rojas, R.: Neural Networks: A Systematic Introduction. Springer, Heidelberg (1996)
8. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, vol. I. Computer Science Press, Rockville (1988)