Background

Admissibility implies reliability and authenticity. The question at hand is how this is demonstrated for records. Like all scientific analysis, a theory (i.e., a theory of the case) implies testable hypotheses (e.g., John was at the bookstore at or about 7 PM - 7:10 PM on March 12, 2015), which may be tested by refutation (e.g., the record shows that John was at the post office at that time on that date). This paper is about an improvement to the method of [1] for testing hypotheses about such records. Reference [1] identifies relevant prior art and general motivations for the present work, and is incorporated herein by reference. Figure 1 gives a sense of the subject matter. It depicts travel times from l_1 to l_2 as a function of time of day (horizontal). The minimum travel of [1] in this case is a fixed 2 hours, the minimum of all times. There are actually times of day when travel can take only 2 and 2.5 hours, and other times when it takes 5 hours. If you miss a flight, travel time jumps. As you move closer (driving) toward l_2, the travel time drops linearly, but there is an envelope of time and space in which you have to be close enough to l_1 to get back in time to make a flight.

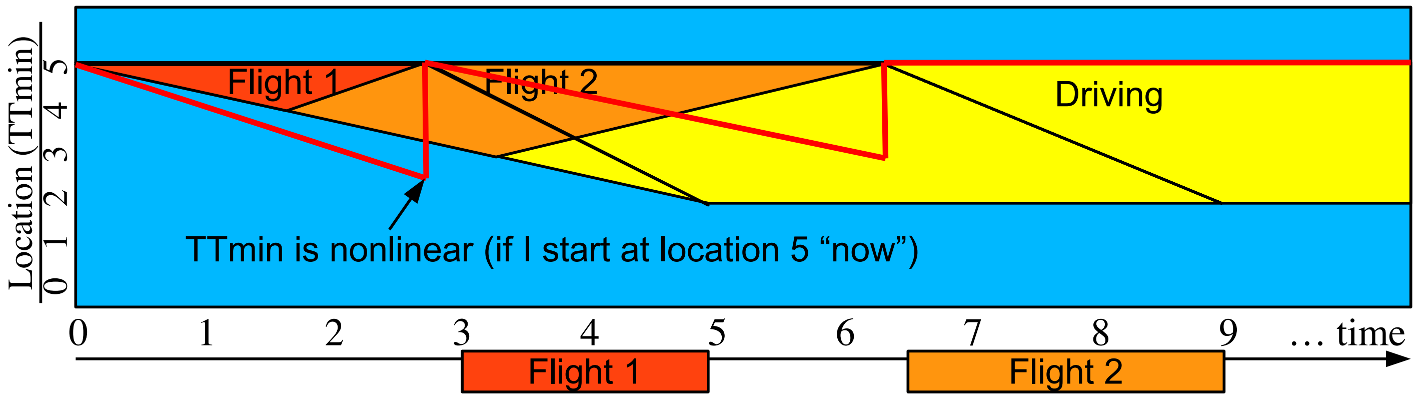
Location (TTmin)
0 1 2 3 4 5
Flight 1
Flight 2
Driving
TTmin is nonlinear (if I start at location 5 "now")
0 1 2 3 4 5 6 7 8 9 … time
Flight 1
Flight 2

Figure 1: A depiction of travel times used for consistency checking.

In assessing the consistency of digital records associated with forensic or diplomatic examination, [1] describes data and algorithms for travel time consistency checking, which is checking that a set of records, each indicating that a certain object was at a certain location at a certain time, is consistent with a set of constraints describing how long it takes to get from one location to another.

The complication that makes this non-trivial is replacement of timestamps with time intervals to represent uncertainty in the recorded times. In the section on future work, [1] lists a variety of generalizations of interest, including:

- constraints on unrecorded travel ("closed spaces")
- non-stationary "locations"
- interactions among objects

This paper discusses some of those along with some further generalizations:

- uncertainty in space analogous to that in time
- constraints on unrecorded travel (more general than closed spaces)
- constraints that depend on time (including non-stationary "locations")
- constraints that depend on the object (but not interactions among objects)

All of the generalizations described here can be combined into a single algorithm which is similar to the original one from [1]. In one case, there is a performance penalty, though the result still has the same order of complexity as the original algorithm. Appendix A shows an outline of the consistency checking algorithm of [1] along with modifications described in this paper. In order to make this paper self contained, we start by trying to summarize the parts of [1] needed to understand the generalizations presented there. The points in the summary are numbered so that they can be referred to later, e.g., as #17.

Primitive data

1. "Locations" are names intended to represent positions in space.

2. "Atomic objects" are names intended to represent objects with bounded locations, those locations being functions of time (i.e., they cannot be in two sufficiently distant places at the same time).

3. Time is assumed to be exactly measurable by a global clock and is represented in the usual ways, either real numbers of seconds since some reference time or something that can be converted to that.

Problem definition

4. The problem input consists of a set of "scheduling constraints" and a "minimal travel time" function, TT.

5. A scheduling constraint represents a record that a given atomic object was at a given location at a time which was not measured exactly on the global clock but by some other mechanism, which is translated into a

time interval as measured by the global clock thought to contain the time at which the recorded observation was made. For example, if the other mechanism recorded 12:44 but it only records time to the minute (no seconds) and is thought to be within 5 minutes of the correct time, the interval would be 12:39:00 - 12:50:00.

6. A scheduling constraint is represented by the name of the atomic object, the name of the location, and the start and end times of the time interval.

7. The minimal travel time function, TT($location_1$,$location_2$) is a function of two locations returning the minimal amount of time it takes any atomic object to get from $location_1$ to $location_2$. This function is required to be zero between any location and itself, greater than zero between any two different locations, and satisfy the triangle inequality. It is not required to be symmetric (think of one way streets).

8. The problem is to determine whether the set of scheduling constraints is consistent with the constraint that no atomic object can get from any first location, $location_1$ to any second location, $location_2$ in less than TT($location_1$,$location_2$).

9. The answer to the problem in #8 is yes IFF there is some "schedule" that both "satisfies" the set of scheduling constraints and "satisfies" (in a different sense) TT.

10. A schedule is a set of appearances, each indicating that a given atomic object was at a given location at a given (exact) time.

11. A schedule, S, satisfies a set of scheduling constraints, C, IFF for every scheduling constraint c in C there is some appearance s in S where the atomic object and location of s are the same as that of c, and the time of s is in the interval of c.

12. A schedule satisfies a minimal travel time function, TT, IFF for every pair of appearances s_1 and s_2 in the schedule with the same atomic object and having locations l_1 and l_2 and times t_1 and t_2 where $t_1 \leq t_2$, $TT(l_1,l_2) \leq t_2-t_1$

Analysis and algorithms

13. The traveling salesman problem is a special case of the travel time consistency problem (just assign the

same interval to all of the scheduling constraints). Therefore we don't expect an efficient algorithm to reliably solve this problem. However, in the cases we expect to encounter, there is relatively little overlap in the time intervals of the scheduling constraints, and the problem can be solved in close to linear (actually n log(n)) time by the algorithms described in [1]. Cases that take more than this are very likely intentionally altered records, and for our purpose it is thought to be normally adequate to simply detect them (which is easy to do in the same time) and limit the extent of their impact on relevant records.

14. Since there are no constraints involving more than one atomic object in this model (i.e., the ignoring “interactions among objects” limitation), the problem can be separated into independent subproblems for different atomic objects. Therefore the first step is to divide the set of scheduling constraints into a separate set for each atomic object. Then each such set will be treated as an independent subproblem.

15. In order to test whether a schedule satisfies TT it is sufficient to test adjacent appearances, due to the triangle inequality.

16. If some schedule satisfies a set of scheduling constraints, C, and also satisfies TT, there is such a schedule with one appearance for each scheduling constraint in C, i.e., there is a bijection between the scheduling constraints and the appearances with each appearance satisfying the corresponding scheduling constraint.

17. An algorithm is shown for testing whether there is a schedule of appearances that satisfies the set of scheduling constraints in a given order and also satisfies TT. This algorithm actually constructs such a schedule if there is one. In fact it constructs the schedule that has each appearance at the earliest possible time. (A similar algorithm constructs the schedule with each appearance at the latest possible time.)

- For the first scheduling constraint in the order, c_1, create an appearance where the time is the start time of c_1.
- As long as there are more scheduling constraints to schedule, do the following to schedule the appearance for the next scheduling constraint, c_{i+1}, using the time just assigned to the appearance for the last one scheduled, c_i:
    - Find the minimal travel time from the location of c_i to the location of c_{i+1}.
    - Add that to the time just scheduled for c_i.
    - If this result is later than the end time of c_{i+1} then there is no schedule (with this order) that satisfies TT.

Otherwise, assign as the time for the appearance for c_{i+1}, the later of the computed result above and the start time of c_{i+1}.

18. Our algorithm searches for an order in which the scheduling constraints can be satisfied while still satisfying TT. Various tricks are used to separate this seemingly n! problem into smaller problems, the first of which is #14 above.

19. A preprocessing phase is used which, in some cases, can reduce the interval in a scheduling constraint on the basis of the time interval of another scheduling constraint in conjunction with the minimal travel times between the two locations. Reducing time intervals tends to improve later search performance, and in some cases can even show that no schedule exists (i.e., the interval is zero). The cost of processing all pairs of scheduling constraints would be quadratic, but this can be reduced to something linear in the number of scheduling constraints. It turns out that there is no benefit in comparing two scheduling constraints with intervals separated by more than the travel time between their locations. In the expected cases, each scheduling constraint has an interval within travel time of only a limited number of other scheduling constraints. This is bounded by the largest number of scheduling constraints overlapping any time interval having as its length the maximum value of TT, which is normally a constant (note the assumption of stationary locations). Or at least it does not depend on the number of scheduling constraints or locations, and is typically a fairly small constant. An algorithm for iterating over pairs of scheduling constraints having time intervals separated by less than some time, T, is:

```
1. create an array A containing the scheduling constraints
2. sort A by start time
3. for i from 1 to A.length do
     for j from i+1 to A.length while A[i].end+T <= A[j].start do
       processpair(A[i],A[j])
```

The time for sorting the constraints is O(n log(n)) where n is the number of constraints and the time for the iteration is proportional to the number of pairs with intervals separated by less than T.

20. When the algorithm above makes some improvement, it may be worth while to run it again to see whether the improved data can be further improved. Unfortunately, it is unknown how many passes may be required to converge, but in practice the answer seems to be two (one forward and one backward).

21. Wherever there is a time interval of length more than the maximum value of TT with no scheduling constraints overlapping the interval, the problem can be separated into one set of scheduling constraints before the interval and another set after the interval, and these two can be solved independently. The algorithm for this:

1. For each scheduling constraint, create a start time record and an end time record (the start time record contains the start time of the scheduling constraint plus the fact that it is a start time, and similarly for the end time record)
2. Create a single array containing all of these records and sort it by time
3. initialize the variable C to 0 (C counts the scheduling constraints in progress)
4. iterate through the array from smallest time to largest, at each step doing:
    - if the next entry is a start record, add 1 to C
    - if the next entry is a stop record, subtract 1 from C (C should never be negative)
    - Whenever C is set (decremented) to 0, compute the difference between the (end) time of the record that caused the decrement to the (start) time of the next record. If this difference is more than the maximum value of TT, the set can be separated by the time interval between the (stop) time of this record and the (start) time of the next record.

22. The overall algorithm first separates the scheduling constraints by atomic object, then for each atomic object does the preprocessing phase (#19) a small number of times (#20), then separates the scheduling constraints by long enough empty time intervals (#21), and finally for each interval containing scheduling constraints does a depth first search for an ordering using an incremental version of #17, where each search node corresponds to an initial segment of a schedule. The operations for getting from one search node to the next are constant time (if we assume evaluating the travel time function is constant time), and in cases of interest the number of search nodes required to find a schedule or determine that there is none is proportional (with the constant being small) to the number of scheduling constraints. Given that we generally consider failure of the search to be linear in the number of scheduling constraints to be a satisfactory result in itself, it is straight forward to simply limit the search by limiting the number of search nodes to a small multiple of the number of scheduling constraints.

23. Some of the algorithms in [1] are constant time operations for moving from one search node to the next, e.g., finding the next scheduling constraint that could possibly be added to the current search node, backtracking to the previous search node. These are independent of the issues in the present paper.

Making TT depend on the atomic object

One way to make the algorithm better model the real world is to make TT a function of the atomic object and the starting and ending locations. For instance, in situations where the atomic objects are people who move by walking or running, different people have different limits on how fast they can move. By failing to distinguish among different people, we treat slow people as if they were fast, since the TT function has to return the minimal travel time over all objects. This fails to recognize as inconsistent the inputs that describe slow people moving unrealistically as fast as fast people. This improvement is very simple from the algorithmic perspective, since our algorithms only use TT in a context in which the (single) relevant atomic object is already identified. We simply replace every occurrence of TT(x,y) with TTA(A,x,y) where A is the atomic object involved in the consistency checking, and, of course, arrange for the new function, TTA, to compute the minimal time required for atomic object A to go from x to y. Note that this generalization combines trivially with all others described below, so it is largely ignored for the rest of the paper.

The pattern of making TT more complex in order to model more complexity in the world recurs in what is to come. When we describe the cost of these algorithms, we always assume that TT is computed in constant time. It seems likely that some of the complexity that we add to TT will end up costing some time. On the other hand we have no basis for expecting that even the original TT function of only two locations was cheap or even for expectations of what this cost might depend on.

Making TT depend on time

Now that we have seen a simple example of extending TT we turn to a more complicated and interesting one, which is making it depend on time. Here are a few examples to convey an idea of what we have in mind:

- Driving from one place to another in rush hour takes longer than at other times when there is less traffic
- If you're at one airport, say LAX, at 1:30 waiting for the next plane to another airport, say SFO, which leaves at 1:45 and then takes 1:10 to get there, you can't get to SFO until 2:55. Your minimal travel time starting at 1:30 is 1:25. However, this minimal time decreases (at a rate of 1) while you wait for the plane. The time it takes to get to SFO at 1:45 is only 1:10.

It turns out that just adding a time argument to TT is not adequate for this generalization. Rather two different travel time functions are needed, one to compute the earliest time allowable to be at a second location after having been at a first location at time t, and another to compute the latest allowable time to be at a first location before being at a second location at time t. The relation between these two functions can be clarified by combining them into a single constraint of the following form:

> If object A is at location l_1 at time t_1, then it is not allowed to be (i.e., it is inconsistent for it to be) at location l_2 during the time between t_2 and t_3.

This could in turn be expressed in terms of a new function,

DisallowedTime(l_1, t_1, l_2) returning the interval (t_2, t_3).

Note that this differs from the TT of [1] in that it combines TT(l_1,l_2) with TT(l_2,l_1). In the model of [1],

DisallowedTime(l_1, t_1, l_2) = (t_1 - TT(l_2,l_1), t_1 + TT(l_1,l_2)).

One of the topics for future work in [1] was non-stationary locations. The idea was that a record indicating a credit card was read at a given time could treat the credit card reader as the location. But the credit card reader might be on a plane. One solution would be to compute the location of the plane at the time the record was made. In some cases that is practical, but in other cases it makes more sense to change our view of what TT means, and say that its arguments are names of sensors (such as credit card readers) which are then free to move. In that case, the travel times between sensors may change because the sensors move, as well as for other reasons above (traffic, waiting for planes, etc.) and all we need to know about the motion of the sensors for purposes of computing schedules and determining consistency is encoded in TT. That is, making TT a function of time is a solution to the problem referred to previously as non-stationary "locations".

Details of travel time as a function of time

Although the DisallowedTime function described above seems a nice theoretical framework for understanding time dependent travel time, it is not most convenient for our computational needs. The TT function is actually used in [1] in two ways:

- if object A is assumed to be at location l_1 at time t_1, what's the earliest later time it could be at l_2
- if object A is assumed to be at location l_1 at time t_1, what's the latest earlier time it could be at l_2

We actually use:

TT1(t_1,l_1,l_2) to compute the amount of time it takes to get to l_2 starting from l_1 at t_1, and

TT2(l_1,l_2,t_2) to compute the minimal amount of time before time t_2 it would be possible to have been at l_1 given that the object was at l_2 at t_2.

We put the arguments of TT2 in a different order to remind the reader that the time argument in TT2 applies to the destination location, which is the second location argument. In both cases, the travel is from the first location argument to the second. Note that TT1 and TT2 are time reversed versions of each other. That is, if we take a history and reverse the signs of all of the times, so that everything happens in reverse order, then the TT1 of that history is the same as the TT2 of the original history and vice versa. In that case, of course the locations are reversed, since the travel is now in the opposite direction. That is, TT1(t_1,l_1,l_2) becomes TT2(l_2,l_1,-t_1) - the argument order is reversed.

We can now begin to see how the problem definition #4 - #12 is affected by making the travel time function depend on time. Corresponding to the original requirements in #7 we now have:

- Both travel time functions must be zero from every location to itself at all times
  .
- Both travel time functions must be positive (or at least non-negative) from every location to every other location at all times.

  The introduction of moving sensors (and also the later section on uncertainty in space) makes it possible for an atomic object to be sensed by two different sensors at the same time, which means that minimal travel time between the two (in both directions) will be zero. In this case we should replace the requirement above with:

  ∀ locations l_1,l_2, times t, if $l_1 \neq l_2$ then TT1(t,l_1,l_2) ≥ 0 and TT2(l_1,l_2,t) ≥ 0. and also,
  ∀ locations l_1,l_2, times t, if either TT1(t,l_1,l_2) = 0 or TT2(l_1,l_2,t) = 0
  then TT1(t,l_1,l_2) = TT1(t,l_2,l_1) = TT2(l_1,l_2,t) = TT2(l_2,l_1,t) = 0

- The triangle inequalities, as before, express the intuition that one way to get from l_1 to l_3 (but now starting at time t_1) is to first go from l_1 to l_2 and then from l_2 to l_3. Therefore the best way should be no worse than that way. The second triangle inequality is analogous for reversed time.

$\forall$ locations l_1,l_2,l_3, times t
$TT1(t,l_1,l_3) \le TT1(t,l_1,l_2) + TT1(t + TT1(t,l_1,l_2),l_2,l_3)$ and
$TT2(l_1,l_3,t) \le TT2(l_2,l_3,t) + TT2(l_1,l_2,t - TT2(l_2,l_3,t))$

Further constraints on travel time appear below (see **TT1-2** and **WAIT**).

The constraint in #8 now becomes that no atomic object can get from any first location, l_1 to any second location, l_2 starting at time t_1 in less than $TT1(t_1,l_1,l_2)$, which we might rephrase as any atomic object at l_1 at time t_1 cannot be at location l_2 at any time between t_1 and $t_1 + TT1(t_1,l_1,l_2)$. Similarly, any object at location l_2 at time t_2 cannot be at location l_1 at any time between $t_2 - TT2(l_1,l_2,t_2)$ and t_2.

Similarly, #12 becomes: A schedule satisfies the minimal travel time functions, TT1 and TT2, IFF for every pair of appearances s_1 and s_2 in the schedule with the same atomic object and having locations l_1 and l_2 and times t_1 and t_2 where $t_1 \le t_2$, $TT1(t_1,l_1,l_2) \le t_2-t_1$ and $TT2(l_1,l_2,t_2) \le t_2-t_1$

We will show below that, after introducing additional constraints on travel time, the two conjuncts in the condition above turn out to be equivalent. To provide some intuition, we first show a sample graph of TT1 and TT2 as a function of time. Driving from l_1 to l_2 in 5 hours is always possible. However, a plane leaving at 3:00 arrives at 5:00, and another slightly slower plane leaving at 6:30 arrives at 9:00.

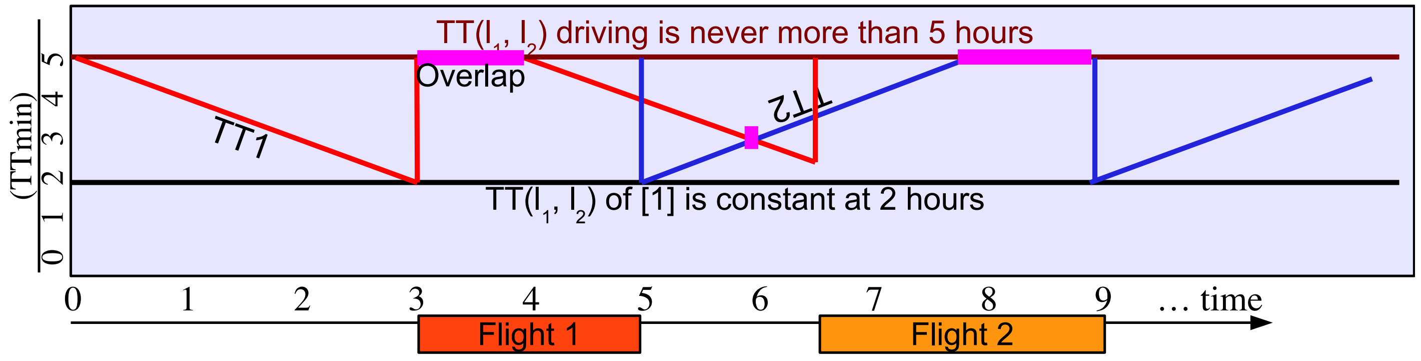

Figure 2: TT, TT1, TT2, and overlaps

What we called $TT(l_1,l_2)$ in [1] is now the minimum value of TT1 over all times, which is the same as the minimum value of TT2 over all times. There is no limit on how far TT1 can rise above that, or how fast. It can increase discontinuously. However it cannot decrease discontinuously and its downward slope can never be more than 1. This graph is certainly not symmetric in time and we might wonder whether we have really found an asymmetry between past and future.

The answer is that the symmetry appears in TT2 which has all the same properties in the negative time direction. When the plane leaves from l_1, the minimal travel time forward takes a sudden jump and then decreases at a rate of 1 until the next plane leaves. The symmetric case is that, when a plane arrives, the latest time at which someone could have left remains the same for a while, meaning that TT2 increases at the rate of 1 and then suddenly drops when the next plane arrives. So viewing time backward we would see the same sudden increases and gradual decreases that we see looking forward in time with TT1.

It seems clear that TT1 and TT2 must be closely related, even though we have not yet shown a single constraint that involves both of them (other than the case where the travel time is zero). We now correct that deficiency. We refer to this as **TT1-2**:

∀ locations l_1,l_2 and times t_1,t_2:
  By definition of TT1, it is possible (consistent) for an object to be both at location l_1 at time t_1, and at location l_2 at time $t_1 + TT1(t_1,l_1,l_2)$.
  This implies that $t_1 + TT1(t_1,l_1,l_2) - TT2(l_1,l_2,t_1 + TT1(t_1,l_1,l_2)) \geq t_1$
  Subtracting t_1 from both sides and adding the TT2 term to both sides we get
    $TT1(t_1,l_1,l_2) \geq TT2(l_1,l_2,t_1 + TT1(t_1,l_1,l_2))$
  Similar reasoning in the reverse direction leads to
    $t_2 - TT2(l_1,l_2,t_2) + TT1(t_2 - TT2(l_1,l_2,t_2),l_1,l_2) \leq t_2$
   or $TT2(l_1,l_2,t_2) \geq TT1(t_2 - TT2(l_1,l_2,t_2),l_1,l_2))$

Examples above of planes leaving and arriving show that the inequalities cannot be replaced with equalities.

TT1-2 requires the symmetry we see between TT1 and TT2 in the graph, but how about the fact that downward movement is limited to a slope of one? This requires another constraint, which we call **WAIT**:

$\forall$ times t_1,t_2 and locations l_1,l_2, if $t_1 < t_2$ then
  $t_1 + TT1(t_1,l_1,l_2) \leq t_2 + TT1(t_2,l_1,l_2)$ and
  $t_2 - TT2(l_1,l_2,t_2) \geq t_1 - TT2(l_1,l_2,t_1)$

That is, you can't get from l_1 to l_2 any earlier by simply leaving later. (We might say you got from l_1 to l_2 "faster" by leaving later if you arrive at the same time, in that the same distance was covered in less time, but you don't arrive any earlier.)

In the same way we can view the triangle inequality as meaning simply that one of the ways to get from l_1 to l_3 is to go first from l_1 to l_2 and then from l_2 to l_3, we can view **WAIT** as meaning that one way to get from l_1 to l_2 starting at time t_1 is to wait at l_1 until some later time, t_2 and then go from l_1 to l_2.

So how does this limit the downward slope to one? Suppose TT1 from l_1 to l_2 at time t is decreasing faster than one, for instance TT1 at time t is 10 time units and one time unit later TT1 has decreased by two time units to be only 8 time units. Then

$t + TT1(t,l_1,l_2) = t + 10$, which is more than $(t + 1) + TT1(t+1,l_1,l_2) = (t + 1) + 8 = T + 9$

This violates **WAIT**.

Although some might consider **WAIT** to be obviously true, we point out at least one case where it is violated. Returning to the use of moving sensors as the location arguments in the travel time functions, suppose we have two traditional immobile sensors as locations l_1 and l_2 and that it takes a person 10 minutes to get from one to the other. Now we add a third sensor, l_3, which is a flying robot that senses a person at l_1, then flies to l_2 in one minute. **WAIT** claims that the person could get from l_1 to l_2 in one minute, since initially the travel time from his location to l_3 is zero, and one minute later the travel time from l_3 to l_2 is zero. All he has to do is "wait" at l_3. The problem is that in reality he cannot do that. Note that this example differs from the sensor being a credit card reader on a plane only in that a person *could* wait at the credit card reader on the plane, and doing so would get him from the point of the flight's departure to its destination at the speed of the credit card reader. If l_3 is equipped with GPS or something similar, it seems better to use this position data as the "location" at which the object was observed. Below we will argue that the opposite choice seems better in the case of the credit card reader on the plane.

We can now prove what we claimed above in relation to #12, that if a schedule satisfies the TT1 part of the constraint then it must also satisfy the TT2 part, and vice versa. Suppose we have two appearances, a_1 in which atomic object A is at location l_1 at time t_1 and a_2 in which atomic object A is at location l_2 at a later (or equal) time t_2.

We use t_2' as a name (or abbreviation) for $t_1 + TT1(t_1,l_1,l_2)$.
Suppose the TT1 part of the constraint is satisfied:
(1) $t_1 + TT1(t_1,l_1,l_2) \leq t_2$
subtituting t_2' for its definition gives
(2) $t_2' \leq t_2$
WAIT says: if $t_1 \leq t_2$ then $t_2 - TT2(l_1,l_2,t_2) \geq t_1 - TT2(l_1,l_2,t_1)$
using t_2' for t_1 in **WAIT**, in combination with (2), we get
(3) $t_2 - TT2(l_1,l_2,t_2) \geq t_2' - TT2(l_1,l_2,t_2')$
TT1-2 says: $t_1 + TT1(t_1,l_1,l_2) - TT2(l_1,l_2,t_1 + TT1(t_1,l_1,l_2)) \geq t_1$
subtituting t_2' for its definition in **TT1-2** gives
(4) $t_2' - TT2(l_1,l_2,t_2') \geq t_1$
combining (3) with (4) gives
(5) $t_2 - TT2(l_1,l_2,t_2) \geq t_2' - TT2(l_1,l_2,t_2') \geq t_1$
using transitivity of ≥ on (5) gives
(6) $t_2 - TT2(l_1,l_2,t_2) \geq t_1$

The last line is the TT2 part of the constraint, so we have proven that if a pair of occurrences is consistent with the TT1 condition then it is consistent with TT2. The symmetry of TT1 and TT2 can be used to construct a similar proof in the other direction.

Impact of time dependent travel time on analysis and algorithms

We now move on to the Analysis and algorithms section, #13 - #23, to find uses of the travel time function and its properties and see how they are affected by the proposed changes.

The triangle inequality was used (#15) to show that only consecutive appearances have to be checked for TT violations. Now we have two different (but symmetric) triangle inequalities. Following [1], suppose there are three appearances, a_1, a_2, a_3 with the same atomic object, and locations and times l_1,t_1 for a_1, l_2,t_2 for a_2 and l_3,t_3 for a_3, with $t_1 \leq t_2 \leq t_3$. (Note that [1] uses < and we now generalize to ≤.) If we assume that the pairs a_1,a_2 and a_2,a_3 both satisfy the TT1 constraint we can prove that the pair a_1,a_3 does also. (The proof is omitted due to space constraints.)

The symmetry between TT1 and TT2 should convince the reader that a corresponding proof works for TT2. Since satisfying TT1 is equivalent to satisfying TT2, we only have to check one of them for each consecutive pair of appearances.

The algorithm for determining whether a given order is consistent with TT (#17) has to be changed in that the travel time function TT changes to TT1 or TT2, depending on whether the earliest or latest appearances are to be found. Of course these functions also need time arguments. The phrase

> Find the minimal travel time from the location of c_i to the location of c_{i+1}.

changes to

> Find the minimal travel time from the location of c_i to the location of c_{i+1} starting at the time just assigned to the appearance for c_i.

The version that computes the schedule with latest possible appearances similarly uses the time just assigned to an appearance as the time argument to TT2.

[1] does not attempt to prove correctness of the algorithm for finding the schedule that puts appearances in a given order and puts each appearance as early as possible (#17). There are two things to prove, first, that a schedule found by the algorithm is correct, and second that when the algorithm fails, there is no such schedule. The intuition for both of these is the same in the TT1 and TT2 version as in the original TT version. The **WAIT**, property is important here, since it allows us to compute the minimal travel time starting from the time assigned to the just scheduled appearance and not have to worry that by scheduling that appearance

for a later time or waiting for a later time to leave, the next appearance might be able to happen earlier. Preprocessing to derive tighter scheduling constraints (#19) again requires changes to replace TT with TT1 in computing later times from earlier ones (using the earlier times for the time argument) and with TT2 in computing earlier times from later ones (using the later times for the time argument). Also, as above, no proof is shown in [1] but the intuition is unaffected by the changes.

The general point that tighter bounds on scheduling constraints are only derived when they are within travel time of each other is still valid, as is the idea of simplifying matters by restricting attention to the maximum TT. However, there is more than one possible interpretation of the idea of maximum TT. Actually, even in [1] the maximum over all minimal travel times was not the only one that could have been used. Another would have been the maximum TT for the particular location argument under consideration. Now that we consider additional arguments for TT, another is the maximum of all minimal travel times for the atomic object under consideration (if travel times depend on the atomic object). Another is the maximum of all minimal travel times for that atomic object starting at the current location at the current time, and so on. All of these lead to the right result. Which is best depends on the cost of computing such maxima and how much work will be saved by using the smaller values that result from maximizing over smaller sets. Both of these are beyond the scope of this paper, since they depend on details of the travel time computation and on the statistical properties of the inputs. The discussion above applies as well to independent processing of subsets of scheduling constraints separated by more than the maximum TT (#21).

Uncertainty in space (nonΔ≠)

The analogy between uncertainty in space and time is not as straight forward as it may at first seem. While we have represented uncertainty in time with time intervals, we have represented space not as analogous positions in a coordinate system (which could conceivably be generalized to regions of space in that coordinate system), but rather names. It should come as no surprise by now that our approach is going to be to adjust the travel time function to reflect uncertainty in location. Before we go into further detail it is worth mentioning that something as simple as replacing a location name with a region in a coordinate space is not particularly useful in terms of computing travel time. For instance, if our location were a credit card reader on a flight, then the uncertainty in its position relative to the earth may be relatively large, but this does not affect its travel time to any other sensor. The travel times between this sensor and others in the plane are unrelated to the uncertainty in the position of the plane. In fact, computing travel time from position with large uncertainty and high velocity would lead us to conclude that any atomic object on a plane could reach any sensor on the plane in nearly zero time, whereas other considerations might indicate that this is not true - there's a drink cart in the way and it's going to take several minutes to get out of the way. For sensors outside the plane, the important data is the time that the flight arrives at its destination and opens its doors, and the travel time from that position (which can be known rather precisely) to the other sensor.

For a better example of uncertainty in space, suppose the sensors are credit card readers in department stores, but stores have a large number of such readers, and the records identify only the store. In essence, then, all of the card readers in one store combine into a single sensor spread over a region of space. This causes the travel time function to violate the triangle inequality. Imagine three such department stores in a row (Sears, Macy's and Gap) where it is possible to get from one to the next quickly, but going from the first to the third takes much longer. Figure 3 shows the non-Δ≠ (left) and Δ≠ (right) analysis and sample results.

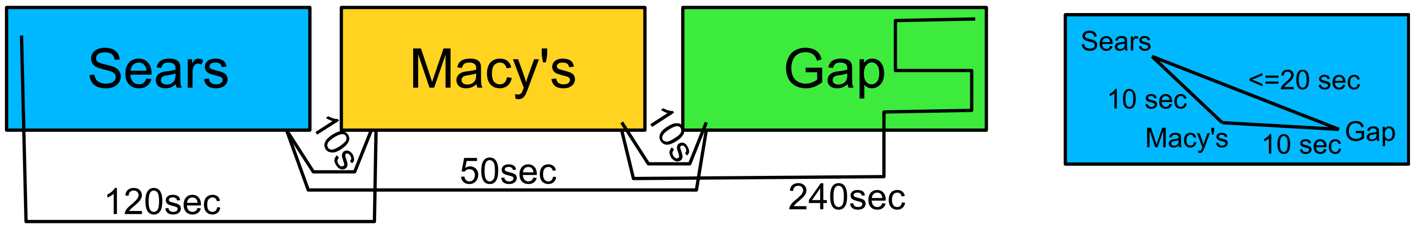

Figure 3: A non-Δ≠ (left) example as opposed to a Δ≠ (right) result

We will refer to travel time functions satisfying the triangle inequality as **Δ≠** (including TT, TT1 and TT2) and those that do not as **nonΔ≠**. In addition to the triangle inequality, uncertainty in space tends to lead to zero travel times between distinct sensors, i.e. it may be possible to be sensed by two different sensors at the same time. (Think in terms of a phone being sensed by two different cell phone towers at the same time.) Fortunately, this issue has been dealt with above.

An interesting example of a limitation on even nonΔ≠ travel times is inability to model a situation where three sensors have pairwise overlapping regions and yet it is impossible to visit all three within a very short time. This is depicted in Figure 4, where different sensor regions (S1, S2, and S3) overlap only in pairs. That would seem to require a new sort of constraint explicitly disallowing visiting three particular sensors within less than a specified amount of time. Naturally we can then construct examples of larger sizes, n, in which it is impossible to visit all n sensors quickly even though all subsets of size n-1 can be visited quickly, leading to new forms of constraints for each value of n.

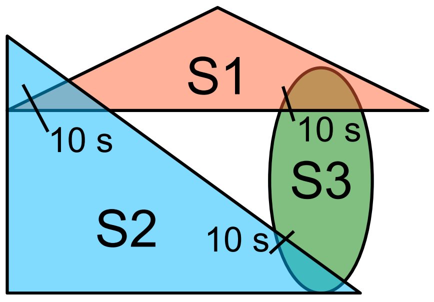

Figure 4: Overlapping sensor regions

Impact of nonΔ≠ travel time on algorithms

With nonΔ≠ travel time it is no longer true that travel times only have to be checked for adjacent pairs of appearances (#15). The algorithm for generating a schedule with all appearances as early as possible for a particular order (#17) no longer works, since, after going from l_1 to l_2, the minimal travel time from l_2 to l_3 might get to l_3 before the minimal travel time from l_1 to l_3. The obvious way to fix this would change a linear algorithm into a quadratic one: compute the earliest possible time of an appearance, a_{n+1}, being added to the end of a schedule, $a_1 \ldots a_n$, as the maximum over all i such that $1 \le i \le n$ of the earliest time the object can get from the location of a_i to that of a_{n+1}. If a maximum TT can be computed efficiently and the value is not too high, then this can still be used to limit the search back through previous appearances. Since the number of appearances in a time interval can be expected to be proportional to the length of the interval, the cost is then multiplied by something proportional to the maximum TT. This is the same trick that is used in preprocessing scheduling constraints (#19). The preprocessing of scheduling constraints (#19) does not rely on Δ≠ travel time and is not affected. The search for an acceptable order (in #22) is an incremental version of the earliest possible schedule algorithm (#17) and pays a similar cost. [1] regarded that cost as linear in the number of scheduling constraints.

Closed spaces and Involuntary recording (CTT)

One of the topics for future work mentioned in [1] was "closed spaces". These are regions of space with a boundary separating the "inside" from the "outside" where an atomic object cannot get between the inside and outside without being sensed and recorded on the boundary in a form which our program will see as a scheduling constraint. In some cases the requirement applies only to one direction of travel, such as entering the secure area of an airport, whereas exit is not recorded. One of the theorems in [1] was that any subset of a consistent set of scheduling constraints was still consistent. This is no longer true with closed spaces. One might view the change as being that in [1] it is tacitly assumed that an atomic object could visit a location without creating a scheduling constraint. Now we require a scheduling constraint when an atomic object visits a location on the boundary. The definition of a schedule satisfying a set of scheduling constraints (#11), must change to require every appearance in the schedule to satisfy at least one of the scheduling constraints - a set of only two scheduling constraints involving the same atomic object on opposite sides of the boundary is not to be considered consistent due to the possibility of creating a schedule with a third appearance on the boundary. Given that we will eventually create a schedule with a bijection between the appearances and the scheduling constraints, it seems simplest to just require this bijection in #11.

What we now describe (and have implemented) is a generalization of closed space constraints which we call involuntary recording. The idea is that just being at a location (or getting close enough to a sensor) will generate a record. In order to avoid generating a record you might have to go out of your way and that might increase the travel time. We use the term **CTT**, standing for "covert travel time", for a new function that computes the minimal time required to get from one location to another without being recorded anywhere along the way. The only constraint on CTT is that it must be at least as large as TT on the same arguments. We also define, analogous to #12, that a schedule satisfies a minimal covert travel time function, CTT, IFF for every pair of *consecutive* appearances s_1 and s_2 in the schedule with the same atomic object and having locations l_1 and l_2 and times t_1 and t_2 where $t_1 \leq t_2$, $CTT(l_1,l_2) \leq t_2-t_1$.[1]

1 A small technical problem arises when we combine closed spaces with other extensions above that allow an atomic object to be at two different "locations" at the same time. Suppose that object A is at location l_1 at time t_1 then at later time t_2 is at both l_2 and l_3. For purposes of TT this presented no problem. Either location

In order to express a closed space constraint we can make the value of CTT between forbidden consecutive pairs larger than the time difference between any two scheduling constraints under consideration. In order to see that CTT is strictly more general than closed spaces, consider a situation where a building has an emergency exit which automatically records those who exit, and a regular exit which does not. The only path with a small travel time from a sensor just inside the emergency exit and another just outside requires going through the emergency exit and being recorded there. In order to avoid being recorded it is necessary to take the longer route through the regular exit. This is not a closed space but it is modeled by CTT.

CTT functions tend to be non$\Delta\neq$ since some involuntary sensor is likely to be on the fastest path between some pair of other sensors. One might therefore expect involuntary recording to require the same algorithmic changes described above for uncertainty in space. However, the CTT constraint only applies to consecutive appearances. As long as TT is $\Delta\neq$ the additional tests are not needed. Analogous to TT, CTT can be made dependent on time by defining CTT1 and CTT2. This allows modeling such things as closed spaces being created or temporary suspension of exit recording due to a fire alarm and sensors moving into and out of closed spaces. As mentioned above, such changes must occur at a specific time since travel time is a function of time. CTT in conjunction with dependence on atomic object can forbid certain atomic objects from reaching certain sensors, and in conjunction with time dependence such restrictions can also be temporary. Of course, all of these complicated behaviors require complicated CTT functions. The **WAIT** and **TT1-2** constraints apply also to the time dependent versions of CTT1 and CTT2.

Impact of CTT on algorithms

The main change required to the algorithm from [1] for CTT is that in #17 and in the incremental version in

could be viewed as the next appearance and the algorithms still work. However now we are supposed to compare t_2-t_1 with CTT, which might be different for l_2 and l_3, and in particular CTT could be satisfied for one and not the other. A number of solutions are possible, such as regarding the order of the appearances to be additional information not simply derived from their times (but still required to at least be consistent with the times). In that case it is really possible for the schedule with l_2 first to be consistent while the schedule with l_3 first is not. Alternatively, one could place additional constraints on CTT to ensure that either both schedules are consistent or neither is.

#22, CTT has to be used instead of TT to compute the earliest next time an atomic object could be at the next location (or the latest previous time at the previous location). Since CTT is at least as large as TT, use of CTT in consecutive appearances still guarantees that the consecutive appearances will satisfy TT. [2]

In the preprocessing phase (#19) we generally do not know whether one scheduling constraint will end up being scheduled immediately after another, so in general use of CTT is not justified in deriving smaller time intervals. However, there are some cases where start and end times can be shown to require that two scheduling constraints must be represented by consecutive appearances, and in those cases more accurate results can be obtained by using CTT.

The last topic is how CTT affects the use of maximum TT in limiting search. Specifically, if maximum TT had to be replaced by maximum CTT, our arguments about linearity would be ruined, since in some cases, like closed spaces, the maximum CTT is effectively infinite.

There are three different uses of the maximum TT. One was added above in response to non$\Delta\neq$. This was used to check TT between a newly added appearance and the non-consecutive previous appearances. Since CTT is only relevant to consecutive appearances, this case requires only maximum TT, not CTT.

Another case is in preprocessing scheduling constraints. In that case we concluded that CTT should be used only if it can be proven that the two scheduling constraints had to be scheduled adjacent to each other. This is similar to the non$\Delta\neq$ case. The first scheduling constraint being compared to a given scheduling constraint might be compared with CTT, but after that we can stop when the time differences exceed the maximum TT.

The final and most interesting case is separation of a set of scheduling constraints into multiple independent

2 It turns out that there is still a good use for the old TT function, even in this case where the time of the next appearance is computed by CTT. A somewhat subtle point in [1] (not yet mentioned in this paper) is that if the search for a schedule discovers that the schedule so far under consideration, a_1 ... a_n, cannot be extended with an appearance for some remaining scheduling constraint due to the fact that the earliest possible arrival from a_n would be too late (actually this applies if there is *any* such remaining scheduling constraint), then the schedule so far under consideration cannot be extended to a satisfactory schedule at all - it's time to backtrack from a_n. This is not true of CTT, however. Therefore, even though the schedule is built by adding CTT from a_n to some proposed a_{n+1}, if that time turns out to be too late to arrive at a_{n+1}, it is worth while to check whether TT would also be too late, since that would justify backtracking from a_n.

sets on the basis of a gap of more than the maximum TT (#21). Here it appears that CTT actually is relevant, since a set of two scheduling constraints on opposite sides of a boundary violate the constraint even if they are separated by more than the maximum TT. Of course it would be safe to use the maximum CTT instead of the maximum TT in separating the set of scheduling constraints, but in the case of a simple closed space this would prevent any separation at all.

Fortunately, there is a fairly good solution, at least under the assumption that scheduling constraints are mostly linearly ordered. We first tentatively separate a set of scheduling constraints into two sets, S1 and S2 on the basis of a gap of maximum TT. We can then find all of the scheduling constraints that could possibly be scheduled last in S1 on the basis of some simple and easy to compute test, e.g., all those with end times at least as late as the largest start time. Call that set S1end. Similarly find the set that could possibly be scheduled first in S2 and call that set S2start. We expect often there will be only one of each, and rarely very many. Now for each pair of scheduling constraints where the first is in S1end and the second is in S2start, test on the basis of CTT whether that pair is consistent. There are three possible outcomes. If all pairs are consistent then we can go ahead and separate the two sets. If no pairs are consistent then we know that the entire original set of scheduling constraints is inconsistent because there is no way of getting from S1 to S2. In the remaining case we can simply not separate the set at that gap and allow the normal search mechanism to resolve the problem. Naturally, we hope this will be the unusual case.
In fact there seems to be no shortage of potential optimizations, and which are worth while depends on the particular problem inputs. This seems to be a characteristic shared by all NP-complete problems. We have been working on the assumption that the order of appearances is mostly determined by the start time of one scheduling constraint being later than the end time of another, and this tends to suggest that most optimizations will not be worth while, especially any that increase overall complexity.

Conclusion

This paper describes a series of improvements to [1]. Most of the problems described there as future work are solved here. Furthermore, somewhat surprisingly, these improvements have not been at very high computational cost, even when they are all combined into a single algorithm. Only one seems to increase the cost significantly at all. Of course, this analysis does not include the cost of computing (and for that matter developing) the appropriate versions of travel time functions to be used in the algorithms reported here.

References:

[1]Fred Cohen, Don Cohen: “Time and space interval record schedule consistency analysis for atomic items without interactions in open spaces with stationary locations.” Computers & Security 45: 293-304 (2014)

Appendix A: outline of consistency checking algorithm

Sections with red labels are added or modified parts of the original algorithm. Note that dependence of travel time on atomic object and time are not shown explicitly since they affect all uses of travel time.

```
I. group scheduling constraints by atomic object
II. for each atomic object, O, process the scheduling constraints for O:
   A. preprocess the set of scheduling constraints:
      alternate between forward and backward in time passes until
      a pass through the set makes no changes
        1. sort by start time, compare each scheduling constraint to
           those after it until time difference > max of min TT
        2. CTT:
           in cases where times require adjacent appearances, use CTT instead of TT
   B. separate the set of scheduling constraints into independent subsets
        1. sort by time, find gaps of inactivity > max of min TT
        2. CTT:
           determine whether adjacent sets must, must not or might satisfy CTT
           must not => inconsistent
           must => leave the two separated
           might => keep them joined together
   C. search for a consistent schedule using depth first search; expand-node does:
        1. find remaining scheduling constraints that could be next
           on the basis of start/end times
        2. for each candidate (until success or resource limit) create a new node
           a. CTT:
             reject the node if it violates CTT (for original algorithm CTT = TT)
           b. nonΔ≠ TT:
             if TT is nonΔ≠ then also check TT against earlier
             appearances back to max of min TT
```