Temporal Constraint Satisfaction

16.413 Project

Amy Brzezinski and Michael Andrew Park

December 9, 2005

1 Introduction

Temporal constraint satisfaction problems (TCSPs) arise from the challenge of scheduling a set of activities constrained to different time intervals. The easiest way to express these activates is by defining their beginning and end as events. A TCSP allows events to be related by temporal distance constrained to single or multiple time intervals. Single events can also be restricted to single or multiple absolute time intervals. The presence of multiple intervals in constraints is what makes TCSPs interesting and potentially expensive to solve.

A TCSP can be formally defined as consisting of a "set of variables...having continuous domains [with] each variable [representing] a time point."[1] TCSP variables are called events and are defined as an instant in time. Events are constrained through unary constraints and a network of binary constraints. Unary constraints restrict the domain of the event to one or more possible intervals. Binary constraints relate the possible temporal distances between two events. Any given event can have multiple unary and binary constraints related to it, each of which can consist of multiple time intervals. TCSPs can be applied to general scheduling and planning problems in a variety of fields including manufacturing and operations.

To solve a TCSP effectively, the problem must divided into multiple smaller simple temporal problems (STPs). A STP can be formally defined as a "TCSP in which all constraints specify a single [continuous] interval."[1] This reduced problem amounts to solving a set of linear inequalities that relate events to each other. The construction of a STP by assigning a single interval to each constraint is performed by a simple depth-first backtracking search algorithm. The original TCSP solution can be constructed by completely enumerating the individual STPs, solving each of these STPs, and compiling the union of the STP solutions. The combination of these intervals is the the complete TCSP solution.

An Individual STP can be solved by the Floyd-Warshall's all-pairs-shortest-paths algorithm by expressing its constraints as a weighted directed graph. The events become the nodes and the constraints become the edge weights. The STPs consistency is determined by detecting negative cycles in the STPs graph

representation. Floyd-Warshall runs in polynomial time and computes minimal network of the solution, which is used to express the temporal relation between all of the events.

Path Consistency algorithms are also very popular in solving STPs or simplifying TCSPs (such as PC-1 and PC-2 as described by Dechter[1]). However, these algorithms do not necessarily converge to a minimal network solution and can have difficulty detecting negative cycles. The Directional Path Consistency (DPC) algorithm is similar to Path Consistency algorithms, except it is a weaker and less constraining version. A drawback of DPC is that it will not always detect inconsistencies in the network. The Floyd-Warshall all-pairs-shortest-paths algorithm is an extremely simple algorithm which excels in its ability to solve STP components of the TCSP. As compared to the path consistency algorithms, the Floyd-Warshall algorithm is the most reliable in solving STPs, as it detects inconsistencies and guarantees a minimal network solution.

The backtracking depth-first search algorithm combined with the Floyd-Warshall all-pairs-shortest-path algorithm is the preferred solution method for solving a TCSP and its resulting STPs. Floyd-Warshall is used for solving individual STPs primarily because of its ability to detect negative cycles, determine consistency, and find minimal network solutions. The ability of this algorithm to perform these activities provides for robust and complete solutions to TCSPs.

2 Problem Description

The problem description begins with the motivation for this work. Then, the problem formulation is decomposed into inputs, outputs, and the definition of a correct solution.

2.1 Motivation for Problem

In our increasingly fast-paced world, scheduling is becoming an important issue. This is especially true in industry, where multiple activities need to be performed under tight resource and time constraints. Temporal constraint satisfaction problems (TCSPs) thus provide solutions on how to best schedule a network of activities that are under absolute and relative temporal constraints. It provides a minimal solution in the form of a temporally consistent schedule for the activity network, thus insuring activity plan productivity.

2.2 Formal Problem Statement

The problem formulation consists of inputs, outputs, and the definition of a correct solution.

2.2.1 Inputs

The inputs into a TCSP problem consist of variables, called events, and constraints, either unary or binary. An event is defined as a singular point in time

considered to have zero duration. The set of events in a TCSP is given by $\{X_1, X_2, \ldots, X_n\}$. An event is constrained individually by a unary constraint C_i which restricts the event's domain to a given set of intervals. A constraint between two events is called a binary constraint $C_{i,j}$. Binary constraints reflect the allowable values for the temporal distance between two events and are often used to express the duration of a proposition. It should be noted that a constraint $C_{i,j}$ is equivalent to constraint $C_{j,i}$ with the interval bounds negated. For simplicity, all unary constraints will eventually be converted into binary constraints between the event and an additional event that represents a fixed epoch. Constraints that involve more than two variables, such as tertiary constraints, can be broken down into binary constraints.

A proposition is a truth that holds for the time period bounded by beginning and end events. Activities are described as propositions in a TCSP. Although the entities that involve duration (propositions and activities) are often of primary interest, it is easier to express and solve a TCSP containing only events and constraints. Fortunately, propositions can be readily converted into the start event, the end event, and a binary constraint to express duration.

Figure 1 shows a portion of a two commuter example problem to be fully introduced in the next section. In this portion, the events (variables) of the problem are $\{X_1 =$ John Leaves Home, $X_2 =$ John Arrives at Work$\}$.

A unary constraint C_1 constrains X_1 to the interval [7:10, 7:20]. The binary constraint $C_{1,2}$ pertains to John's travel time from home to work. It expresses the temporal distance between X_1 (John leaves home) and X_2 (John Arrives at work). Formally, this binary constraint can be expressed as $C_{1,2} = \{$30-40 min by car, 60-∞ min by bus$\}$. Notice that this binary constraint imposes two possible intervals relating events X_1 and X_2. An example of a proposition in this problem is "John is commuting to work", a truth for the time period bounded by the events "John Leaves Home" and "John Arrives at Work."

2.2.2 Outputs

Outputs from a TCSP can consist of timelines, representations that show intervals when events or propositions could occur relative to each other. These outputs can also be formatted into schedules, indicating specific times when events or propositions can occur. These timelines or schedules could display either events (singular points in time), or propositions/activities (intervals during which a truth holds for a bounded temporal period). An example of a representation that could be an output of a TCSP is a Gantt chart (see Figure 2 for an example), which shows a timeline or schedule of activities (tasks).

2.2.3 Specification of a Correct Solution

In the formulation of a correct solution for a TCSP, all variable temporal constraints, both unary and binary, must be fulfilled. Additionally, resource constraints related to events or propositions must be observed in a correct solution. A TCSP can contain consistent and inconsistent STPs. A correct TCSP solution

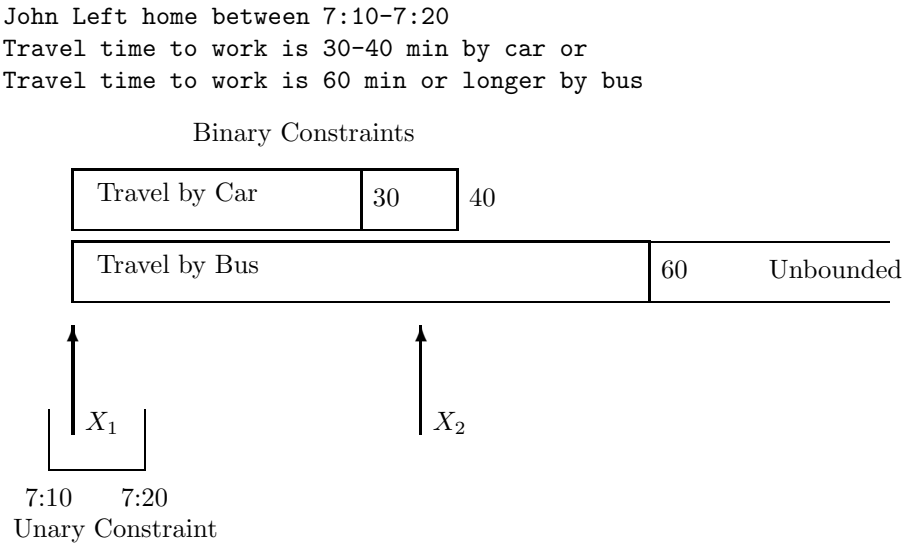


Figure 1: John Commuting.

insures that all the STPs that comprise it are consistent, with all inconsistent STPs pruned. Each consistent solution provides all events assigned to intervals. From these assignments, proposition intervals may be found.

3 Educational Examples

The following examples are provided to explicitly illustrate the steps required to solve an TCSP. The first example is a commuter problem that permits TCSP calculation by hand. The second example is a parameterized airline scheduling problem that can be made large, to test algorithm performance, or small, to also permit hand calculation for algorithm verification.

3.1 Two Commuters

This example is taken from Dechter[1]. The goal of the problem is to determine if the information about John and Fred listed in Table 1 is consistent, and if so, to determine when John and Fred left home and arrived at work.

Formally, the problem can be defined as a set of variables and a network of unary and binary constraints.

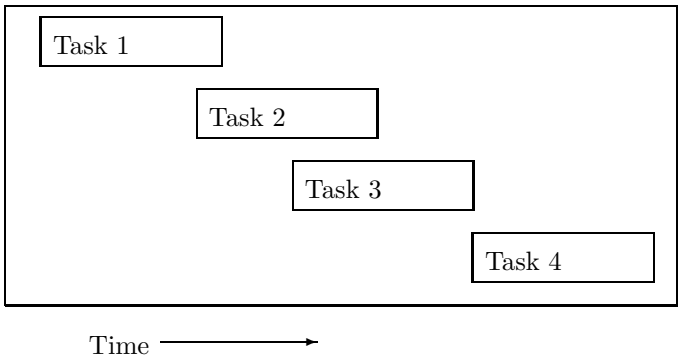


Figure 2: An Example of a Gantt Chart.

Table 1: Two Commuter Example Story.

John:	Travel time is 30–40 min by car or 60–∞ min by bus
Fred:	Travel time is 20–30 min by car or 40–50 min by carpool
John:	Left home between 7:10–7:20
Fred:	Arrived at work 8:00–8:10
John:	Arrived at work 10–20 min after Fred left home

Events:	X_1 = John Leaves Home
	X_2 = John Arrives at Work
	X_3 = Fred Leaves Home
	X_4 = Fred Arrives at Work
Unary Constraints:	C_1 = [7:10,7:20]
	C_4 = [8:00,8:10]
Binary Constraints:	$C_{1,2}$ = [30,40],[60,∞]
	$C_{3,4}$ = [20,30],[40,50]
	$C_{3,2}$ = [10,20]

The unary constraints are given in absolute time, while the binary constraints are given in minutes. After feasibility has been established, additional queries can be posed, such as, "Could John and Fred have taken the bus and carpool, respectively?"

3.2 Airline Scheduling

A large, non-trivial example of a TCSP is scheduling planes flying to and from different airports. Depending on how many planes are being scheduled, how many flights they take, and how many airports are involved, this problem can

become extremely complex. To simplify the problem and allow TCSP solution performance evaluation, an easier formulation consists of n planes flying from n different airports to a hub, and then each plane leaving the hub to return back to its original airport. Figure 3 shows a diagram of such an example.

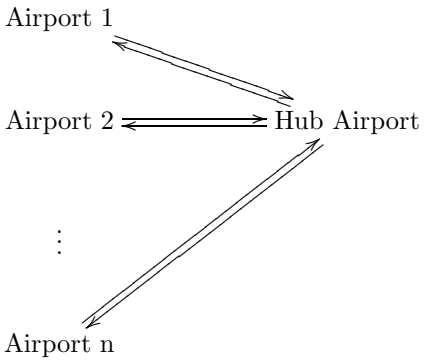


Figure 3: Parameterized Aircraft Scheduling Routes.

Each airplane i has four events associated with it: $X = \{$airplane i leaves airport i, airplane i arrives hub, airplane i departs hub, airplane i arrives airport $i\}$

There are two types of unary constraints in this problem. The first unary constraint restricts all flights to occur between 6:00 am and 11:59 pm. Thus, for each airplane, all four airplane events have the constraint $C = [6{:}00, 11{:}59]$. The second unary constraint restricts when flights can arrive to and leave from the hub. Thus, these unary constraints restrict the events {airplane i arrives hub, airplane i departs hub} for each airplane. In this example problem, the number of intervals on this constraint is set by the user. The length of each interval is two hours long, with the first interval beginning at 6:00 am. All intervals are separated by an hour. Thus, if there are two intervals during which planes are allowed to enter and leave the hub, then the constraint intervals are [6:00 am, 8:00 am] and [9:00 am, 11:00 am].

A variety of binary constraints exist for events pertaining to one plane and between planes. The first binary constraint, flight time, constraints the set of events {airplane i leaves airport i, airplane i arrives hub} and {airplane i departs hub, airplane i arrives airport i} for each airplane. In this example problem, the flight time from any airport to the hub is [60, 90] minutes, while the time from the hub to any airport is [70, 120] minutes.

The second binary constraint, passenger layover time, constrains each plane's arrival at the hub with every other plane's departure from the hub. It ensures that all passengers on one plane may transfer to another to go to a different airport destination. For example, an individual on airplane 1, upon arrival at the hub, may want to switch to airplane 5 to go to airport 5. Thus, airplane 1 needs to arrive at the hub before airplane 5 departs the hub so the individual

can make their connection. All passenger layover time constraints constrain the two events {airplane i arrives hub, airplane j departs hub} to the interval [60, 120] minutes for all i's and j's

The third binary constraint, takeoff and landing spacing time, insures that all planes do not take off or land at once. In this problem, we assume that takeoffs and landings occur on separate runways, a common practice at airports. For simplicity, all airplanes take off and land in numerical order (airplane 1, airplane 2, etc.). Thus, the sets of events for $i < j$ {airplane i arrives hub, airplane j arrives hub} and {airplane i departs hub, airplane j departs hub} are constrained by the interval [5,∞] minutes. The minimum five minute spacing insures that all wake turbulence is dissipated before another airplane takes off or lands.

The fourth binary constraint, airplane turnover and refueling time, constrains the events {airplane i arrives hub, airplane i departs hub} to the interval [45,60] minutes for each airplane. This insures that planes can be serviced between flights.

The direct user inputs into this problem are the number of planes in the network and the number of two hour intervals during which planes can arrive and depart from the hub. The solution output consists of a TCSP network with time intervals relating airplane events to each other and to the reference time of 6:00 am. The performance of the TCSP solution method, as pertains to this example, is discussed in Section 7.

4 Application Programming Interface (API)

The Application Programming Interface (API) for this project is implemented in Java. A stripped down version of the main classes for this project is provided in the report for completeness, but the the project JavaDoc documentation should be consulted for more complete information. Two helper classes, `Event` (Figure 4) and `Interval`(Figure 5), support the main TCSP Java class, `TempCSP` (Figure 6).

```
public class Event {
    /* Constructor sets the description string. */
    public Event( String description )

    /* returns description string. */
    public String description( )
}
```

Figure 4: `Event` Java Interface.

```
public class Interval {
    /* A constant to specify unbounded lower intervals */
    public static final double NO_LOWER;

    /* A constant to specify unbounded upper intervals */
    public static final double NO_UPPER;

    /* Constructor that gives an interval a start and end time.*/
    public Interval( double start_time, double end_time )

    /* Constructor that gives an interval a start time,
     * an end time, and a description.*/
    public Interval( double start_time, double end_time,
                     String description )
    /* Returns the start time of the interval */
    public double start_time( )
    /* Returns the end time of the interval */
    public double end_time( )
}
```

Figure 5: `Interval` Java Interface.

```
public class TempCSP {
    /* Constructor that initializes the epoch Event to
     * default name: "Epoch". */
    public TempCSP( )

    /* Constructor that initializes the epoch Event to
     * argument epoch_name. */
    public TempCSP( String epoch_name )

    /* A method that sets a unary constraint to restrict a variable. */
    public TempCSP add_unary_constraint( Event event, Interval interval )

    /* A method that adds a binary constraint between two events. */
    public TempCSP add_binary_constraint( Event a, Interval interval, Event b )

    /* This method is the solution driver for the TempCSP problem */
    public TempCSP solve( )

    /* Returns the current number of unique event names. */
    public int number_of_events( )

    /* Returns a list of event domains (as Interval) in
     * TCSP solution referenced to epoch  */
    public LinkedList event_domains( Event event )

    /* Returns a list of Interval (minimal network) for
     * an event pair in TCSP solution */
    public LinkedList minimal_network( Event a,  Event b )
}
```

Figure 6: `TempCSP` Java Interface.

5 Interface Example

The specification of the two commuter example input description is employed as an example to illustrate the use of the `TempCSP` API. Figure 7 shows this example input.

```
public class TwoCommuters {
    public static void main(String[] args) {
     /* creates a net TCSP and initializes the relative epoch to 7:00 am */
        TempCSP tcsp = new TempCSP("7:00 am");
        /* define events */
        Event john_left_home = new Event("john left home");
        Event john_arrived_work = new Event("john arrived work");
        Event fred_left_home = new Event("fred left home");
        Event fred_arrived_work = new Event("fred arrived work");
        /* constrain travel times */
        Interval john_car_travel_time = new Interval(30,40,"john car");
        tcsp.add_binary_constraint( john_left_home,
                                    john_car_travel_time, john_arrived_work );
        Interval john_bus_travel_time =
            new Interval(60,Interval.NO_UPPER,"john bus");
        tcsp.add_binary_constraint( john_left_home, john_bus_travel_time,
                                    john_arrived_work );
        Interval fred_car_travel_time = new Interval(20,30,"fred car");
        tcsp.add_binary_constraint( fred_left_home, fred_car_travel_time,
                                    fred_arrived_work );
        Interval fred_carpool_travel_time =
            new Interval(40,50,"fred carpool");
        tcsp.add_binary_constraint( fred_left_home, fred_carpool_travel_time,
                                    fred_arrived_work );

        /* absolute time constraints (min past 7 AM) */
        tcsp.add_unary_constraint( john_left_home,new Interval(10,20) );
        tcsp.add_unary_constraint( fred_arrived_work, new Interval(60+00,60+10) );
        /* relative time constraint */
        tcsp.add_binary_constraint( fred_left_home,
                                    new Interval(10,20), john_arrived_work );
        tcsp.solve();
        System.out.println( tcsp.toString() );
    }
}
```

Figure 7: `TwoCommuter` Java Example.

The example java code in Figure 7 is executed in Figure 8. The `TempCSP` package can be downloaded from:
`http://web.mit.edu/mikepark/www/TempCSP/TempCSP.zip`
The solution is listed as a series of pairs of event relations (event a `->` event b) and intervals.

```
athena% unzip TempCSP.zip [or tar xzvf TempCSP.tgz]
[snip...]
athena% cd TempCSP/src
athena% javac TwoCommuters.java [or athena% ant]
athena% java TwoCommuters
solution exists
7:00 am -> john left home
[10.0,20.0][10.0,20.0][10.0,10.0]
7:00 am -> john arrived work
[40.0,60.0][40.0,50.0][70.0,70.0]
7:00 am -> fred left home
[30.0,50.0][20.0,30.0][50.0,50.0]
7:00 am -> fred arrived work
[60.0,70.0][60.0,70.0][70.0,70.0]
john left home -> john arrived work
[30.0,40.0][30.0,40.0][60.0,60.0]
john left home -> fred left home
[10.0,30.0][10.0,20.0][40.0,40.0]
john left home -> fred arrived work
[40.0,60.0][50.0,60.0][60.0,60.0]
john arrived work -> fred left home
[-20.0,-10.0][-20.0,-10.0][-20.0,-20.0]
john arrived work -> fred arrived work
[-0.0,20.0][20.0,30.0][-0.0,0.0]
fred left home -> fred arrived work
[20.0,30.0][40.0,50.0][20.0,20.0]
```

Figure 8: Screen Output from the `TwoCommuter` Java Example on an M.I.T. Athena Linux Machine.

6 Method Description

After the major ideas of the methods are introduced, the implemented algorithm is described through psuedo-code and an explicit example.

6.1 Introduction of Big Ideas

There are no polynomial time algorithms for solving a TCSP. However, STPs can be solved in polynomial time. A systematic method is to reduce the TCSP into exponentially many STPs and then solve each STP. A simple backtracking algorithm creates these STPs by selecting and assigning one interval for each constraint in the problem in a depth-first search manner. The method of breaking down the TCSP into smaller STP problems allows for the total problem to be solved simply and systematically.

Individual STPs are solved using Floyd-Warshall's all-pairs-shortest-path algorithm, which is nested within the backtracking algorithm. Floyd-Warshall examines an STP's constraints until consistent intervals are found between all events, or an inconstancy is found. Floyd-Warshall is initiated by constructing a distance graph, where the constraint intervals are converted into weighted, directed edges connecting events represented as nodes. The algorithm iterates through the different paths in the directed edge graph until all shortest paths are found. These sets of shortest paths can indicate inconsistent STP constraints or be be converted to intervals to compile solutions to the parent TCSP.

Solvable STPs will produce minimal network solutions in the form of intervals assigned to events consistent with all network constraints. Some STPs will not be solvable, in that events will have interval assignments that are not consistent with network constraints. All solvable STP solutions can be combined in union to find the minimal network solution for the TCSP. The advantage of using the Floyd-Warshall algorithm to solve STPs is that it determines the STPs consistency (by detecting negative cycles), computes the minimal domains and network for the problem, and runs in polynomial $O(n^3)$ time. After an STP is either solved or found to be unsolvable, the backtracking algorithm re-assigns different intervals to one or more of the constraints in order to generate a new STP.

6.2 Algorithm Description with Pseudo-code

A TCSP event X_i may be constrained by multiple unary or binary constraints. Figure 9 shows how STPs are generated from a TCSP. Recall that a STP is a TCSP with at most one unary constraint restricting the domain of each event and at most one binary constraint relating events. Once a STP has been constructed it can be solved by satisfying a set of linear inequalities on the domains of the events. These linear inequalities can be solved by a simplex method, but the structure of the problem permits more efficient algorithms.

There are two keys to efficient solution of TCSP. The first is the selection of reduced intervals when the STP is formed, because the number of STPs that

- Reduce the TCSP to an STP
  - Form directed graph from STP intervals.
    - Each event X_i is represented as a node
    - An additional event X_0 is added to represent absolute time and all unary constraints are converted to binary constraints with respect to X_0
    - All binary constraints are converted into directed graph edges with the upper interval limit being a weight for a directed edge and the lower bound being a negative weight for a directed edge in the reverse direction.
  - Compute the minimum distances
    - See Figure 10 for Floyd-Warshall's All-pairs-shortest-paths algorithm description
  - If no negative cycles are detected, transform minimum distances to constraint intervals $[-d(j,i), d(i,j)]$.
- Find the union of intervals from all consistent STP to form the TCSP solution

Figure 9: TCSP Solution Pseudo-code.

can be enumerated from a single TCSP grows exponentially with the number of multiple constraints. A simple backtracking search is employed, but more sophisticated methods, such as backjumping, variable ordering, value ordering, and learning schemes can reduce execution time of the search for consistent STPs.[1] The second is the quick solution of the STP, where it is advantageous to detect inconsistent constraints as soon as possible so that computational resources are not wasted on refining an inconsistent solution. Floyd-Warshall's All-pairs-shortest-paths algorithm, shown in Figure 10, is used to determine consistency by detecting negative cycles, and finds the minimal STP network if the constraints are consistent. The matrix $d(i,j)$ is the shortest path between the variables i and j on the distance-directed graph, which is found in order to generate the STP minimal network solution.

6.3 Walked Through Example

Recall the example from Section 3.1 about the two commuters, John and Fred. Figure 11 shows this problem mapped as a graph of the relations between the events. The variables describing these events are defined in Table 2. Events are the nodes of the graph, while the binary constraints between them are the edges. All unary constraints are converted to binary constraints in forming the graph. This new binary constraints relate the event to an arbitrary absolute time. In our example, the variable X_0 is added in order to convert the unary constraints

- Create matrix d sized (number of events)×(number of events) to hold the shortest paths between nodes.
- Initialize all d entries to ∞, except set the diagonal of d (the distance between a node and itself) to 0
- For each constraint
  - set d(event1,event2) = (interval end)
  - set d(event2,event1) = -(interval start)
- Find shorter paths from all existing shortest paths

```
for k = 1 to number of events
  for i = 1 to number of events
    for j = 1 to number of events
      d(i,j) = min( d(i,j), d(i,k) + d(k,j) )
      if ( i==j and d(i,j) < 0.0 ) return inconsistent
```

Figure 10: Floyd-Warshall's All-Pairs-Shortest-Paths Algorithm Pseudo-code.

Table 2: Variable Description for Two Commuter Example

X_0	The absolute time of 7:00 am
X_1	The absolute time John left for work
X_2	The absolute time John arrived at work
X_3	The absolute time Fred left for work
X_4	The absolute time Fred arrived at work

associated with X_1 and X_4 to binary constraints. Some of the constraints (edges) between events have more than one interval, making this problem a TCSP. The first step for our algorithm to solve this problem is to break it down into an STP by applying the backtracking algorithm which assigns one interval to one constraint.

The constraints $C_{1,2}$ and $C_{3,4}$ are the only two constraints that have multiple intervals; the other constraints will always be assigned to their one interval for all STPs generated. Figure 12 shows the four possible STPs generated by the backtracking search algorithm's interval assignments. For an example of solving an STP, assume the backtracking algorithm first assigns John to have traveled by car ($C_{1,2}$ = [30,40]) and Fred to have traveled by carpool ($C_{3,4}$ = [40,50]), as in Figure 12(c).

Now that the TCSP has been broken down into an STP, the Floyd-Warshall algorithm can be employed. The first step in solving the STP is to represent

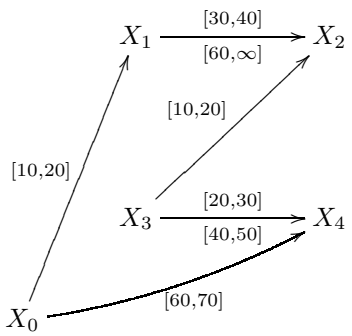


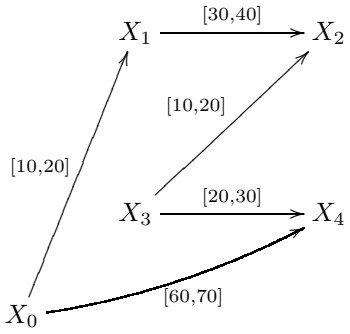
Figure 11: Original Two Commuter TCSP Example.

Figure 12(c) as a directed edge-weighted graph (also known as a distance graph), shown in Figure 13. Each edge interval in Figure 12(c) is converted into a pair of weighted directed edges in Figure 13, with each edge of the pair oriented in opposite directions. The edge weight in the same direction of the interval edge from figure 12(c) indicates the largest duration allowed between the events. The edge weight in the opposite direction of the interval edge from figure 12(c) indicates the shortest duration between events. The combination of these two numbers gives the original interval for the constraint.

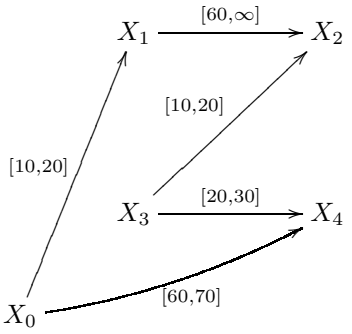
After the distance graph is generated, the algorithm checks to see if the STP is consistent by examining the graph for negative cycles. Negative cycles can be detected by tracing a path through the graph that begins and ends with the same node. If the edge weights of this path sum to a negative number, a negative cycle is present. This negative cycle test needs to be performed for all nodes and all paths on the graph. If a negative cycle has been found, the STP is not consistent and is unsolvable. For example, in checking node X_0 for a negative cycle, from Figure 13 traveling in increasing node number the edge costs added are 20+40-10+50-60 = 40, a positive number indicating this path is not a negative cycle.

The Floyd-Warshall algorithm implicitly detects negative cycles in the process of computing the All-pairs-shortest-paths distances. It starts by assigning incumbent shortest path values for each pair of nodes. All paths between sets of different events are initialized to ∞. Paths from an event to itself are initialized to zero. The actual shortest paths are found by tracing all possible paths between two nodes and adding up the edge costs. For example, in finding the shortest path between nodes X_0 and X_2, there are two possible paths – one that goes through X_1 and one that goes through X_4 and X_3, in that order. The first of these paths (through X_1) gives a path cost of 60, while the other path has a cost of 50. The second path thus is the shortest path length between the two nodes.

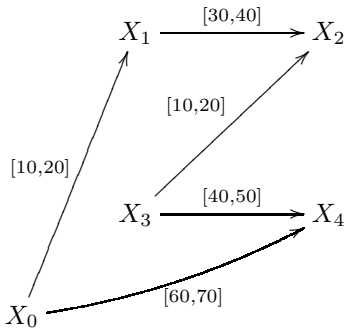
Table 3 shows the lengths of the shortest paths in the distance graph from Figure 13. The row denotes the index of the start node and the column denotes



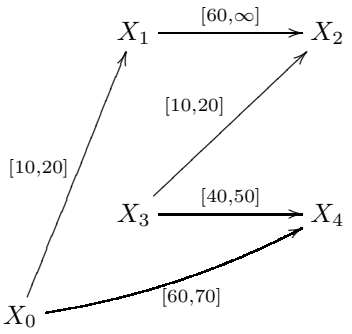
(a) John by Car, Fred by Car.



(b) John by Bus, Fred by Car.



(c) John by Car, Fred by Carpool.



(d) John by Bus, Fred by Carpool.

Figure 12: Four STPs created by enumerating all intervals of the Two Commuter TCSP Example

the end node for a path. Notice that the shortest path from X_0 to X_2 is not the same as the shortest path X_2 to X_0. This STP is consistent, because all diagonal elements are zero. A negative cycle is detected if there is a negative value on the diagonal of Table 3.

From the shortest paths, the minimal network corresponding to the STP's distance graph can be found. This is done by examining each shortest path entry in the table and its diagonal partner. The diagonal partner of an entry is simply the entry mirrored across the diagonal of 0 entries in the table. It also can be expressed as the reverse coordinates of a table entry. For example, if we express an entry in the table as a pair of coordinates (row, column) to designate its tabular location, then the diagonal partner of entry (i, j) is located at (j, i). From our example, (0,1) and (1,0) are diagonal partners.

The minimal network is found for each entry by setting its upper interval

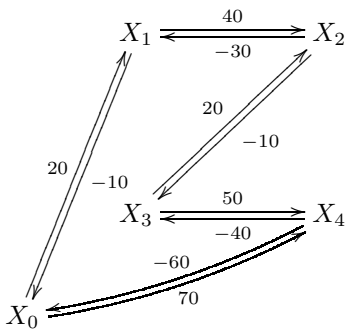


Figure 13: Directed Distance Graph for Two Commuter STP Example (John by Car, Fred by Carpool).

Table 3: Lengths of shortest paths in distance graph

	X_0	X_1	X_2	X_3	X_4
X_0	0	20	50	30	70
X_1	-10	0	40	20	60
X_2	-40	-30	0	-10	30
X_3	-20	-10	20	0	50
X_4	-60	-50	-20	-40	0

bound equal to the corresponding entry in the shortest paths table, and its lower interval bound equal to the opposite sign of its diagonal partner in the shortest paths table. For example, the entry in the minimal network table at (1,3) will have an upper interval bound of 20 (found from entry (1,3) on the shortest paths table), and a lower interval bound of -(-10)=10 (found from (1,3)'s diagonal partner's entry (3,1)). Thus, the minimal network entry for (1,3) will be [10,20]. Each entry in the minimal network table is generated in this manner. Table 4 shows the final completed minimal network for this particular STP. The first row of the table shows the interval of times in which events can occur, relative to the absolute time of 7:00 am. For example, from the entry (0,3), we know that Fred leaves between 7:20 and 7:30 am. One possible variable assignment for the events could be that John leaves for work at 7:10 (X_1) and arrives at 7:50 (X_2), while Fred leaves at 7:30 (X_3) and arrives at 8:10 (X_4).

At this point in solving the TCSP, the backtracking algorithm would backtrack and re-assign different intervals to one or more of the constraints. In this problem, there exist 3 other STPs which need to be solved. However, not all of these STPs are consistent. For example, the STP in which John goes to work by bus ($C_{1,2} = [60,\infty]$) and Fred travels by carpool ($C_{3,4} = [40,50]$ has a negative cycle, indicating the constraints are not consistent and the STP is unsolvable.

After all STPs are solved, the TCSP minimal network is generated by com-

Table 4: Minimal network corresponding to distance graph

	X_0	X_1	X_2	X_3	X_4
X_0	[0]	[10, 20]	[40, 50]	[20, 30]	[60, 70]
X_1	[-20,-10]	[0]	[30, 40]	[10, 20]	[50, 60]
X_2	[-50,-40]	[-40,-30]	[0]	[-20,-10]	[20, 30]
X_3	[-30,-20]	[-20,-10]	[10, 20]	[0]	[40, 50]
X_4	[-70,-60]	[-60,-50]	[-30,-20]	[-50,-40]	[0]

bining all results from the solvable STPs. This is done by taking the union of the STP minimal network solutions. The TCSP minimal network gives the time intervals for the events in which all constraints will hold true. Table 5 shows the minimal network of the whole commuter example. Some entries contain multiple intervals, such as (0,2) and (1,2).

Table 5: Minimal network corresponding to TCSP Commuter Problem

	X_0	X_1	X_2	X_3	X_4
X_0	[0]	[10, 20]	[40, 60] [70]	[20, 50]	[60, 70]
X_1	[-20,-10]	[0]	[30, 40] [60]	[10, 30] [40]	[40, 60]
X_2	[-60,-40] [-70]	[-40,-30] [-60]	[0]	[-20,-10]	[0, 30]
X_3	[-50,-20]	[-30,-10] [-40]	[10, 20]	[0]	[40, 50] [20, 30]
X_4	[-70,-60]	[-60,-50]	[-30, 0]	[-50,-40] [-30,-20]	[0]

The TCSP minimal network solution table relates all events to one another. Row zero of the table describes the relation of all events to the absolute time reference 7:00 am, represented by event X_0. By examining this row, we find that John left home between 7:10 and 7:20 am and arrived at work between 7:40 and 8:00 if he traveled by car and at 8:10 if he traveled by bus. Fred left home between 7:20 and 7:50 and arrived at work between 8:00 and 8:10. Now that this table is presented, answering the previously posed question is left as an exercise for the reader, "Could John and Fred have taken the bus and carpool, respectively?"

7 Implementation Performance

The complexity and execution time of the parameterized Airline Scheduling model described in Section 3.2 is shown in Table 6. The cases where run on a 3.0

Table 6: Airline Scheduling Complexity and Execution Time.

Aircraft	Number of Gate Intervals					
	1			2		
	Total STPs	Consistent STPs	Time (sec)	Total STPs	Consistent STPs	Time (sec)
2	16	8	0.019	81	44	0.027
3	64	8	0.018	729	144	0.074
4	256	16	0.051	6561	608	1.106
5	1024	32	0.259	59049	2496	12.625

GHz P4 Linux desktop with Java(TM) 2 Runtime Environment, Standard Edition version 1.5.0_03. The case is run with the `AirlineSchedulingParametrized` class, which is available in the "src" directory of the TempCSP package.

The first entry in each row of Table 6 represents the number of aircraft n used in each tcsp. The first entry in each column of Table 6 represents the number of intervals m when aircraft can arrive and depart used in each tcsp. The total number of STP that are generated for each problem is given by

$$(3+m)^n. \tag{1}$$

The complexity of this problem is exponential with the number of aircraft. The number of consistant STPs out of this total increases monotonically with the total number of STP, by at a much lower rate. By comparing the execution time to the number of enumerated STPs for the last two rows, An average time of ≈ 0.0002 sec per STP is estimated.

References

[1] Dechter, R., Meiri, I., and Pearl, J., "Temporal Constraint Networks," *Artificial Intelligence*, Vol. 40, pp. 61–95, 1991.

[2] Williams, B. C., "Temporal Plan Execution: Dynamic Scheduling and Simple Temporal Networks," 16.412/6.834J Class Notes, Massachusetts Institute of Technology, 2005.