Magic Loops in Simple Temporal Networks with Uncertainty
Exploiting Structure to Speed Up Dynamic Controllability Checking

Luke Hunsberger

Computer Science Department, Vassar College
Poughkeepsie, NY, USA
hunsberg@cs.vassar.edu

Abstract: A Simple Temporal Network with Uncertainty (STNU) is a structure for representing and reasoning about temporal constraints and uncontrollable-but-bounded temporal intervals called contingent links. An STNU is dynamically controllable (DC) if there exists a strategy for executing its time-points that guarantees that all of the constraints will be satisfied no matter how the durations of the contingent links turn out. The fastest algorithm for checking the dynamic controllability of arbitrary STNUs is based on an analysis of the graphical structure of STNUs. This paper (1) presents a new method for analyzing the graphical structure of STNUs, (2) determines an upper bound on the complexity of certain structures—the indivisible semi-reducible negative loops; (3) presents an algorithm for generating loops—the *magic loops*—whose complexity attains this upper bound; and (4) shows how the upper bound can be exploited to speed up the process of DC-checking for certain networks. Theoretically, the paper deepens our understanding of the structure of STNU graphs. Practically, it demonstrates new ways of exploiting graphical structure to speed up DC checking.

1 BACKGROUND

Agent-based applications invariably involve actions and temporal constraints. Dechter et al. (1991) introduced Simple Temporal Networks (STNs) to facilitate the management of temporal constraints. Vidal and Ghallab (1996) were the first to incorporate actions with uncertain durations into an STN-like framework, and to define a notion of dynamic controllability. Morris et al. (2001) developed the most widely accepted formalization of Simple Temporal Networks with Uncertainty (STNUs) and dynamic controllability. Morris and Muscettola (2005) developed an $O(N^5)$-time algorithm for checking the dynamic controllability of arbitrary STNUs. Morris (2006) presented an $O(N^4)$-time DC-checking algorithm that was based on an analysis of the structure of STNU graphs; it is the fastest DC-checking algorithm reported so far in the literature.

⇒ This paper presents a new way of analyzing the structure of STNU graphs that can be used to speed up DC checking for some networks.

The rest of this section summarizes the definitions and results for STNs, STNUs and dynamic controllability that will be used in the rest of the paper.

1.1 Simple Temporal Networks

Dechter et al. (1991) introduced *Simple Temporal Networks (STNs)* and presented the basic theoretical results for them. An STN is a pair, $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a set of time-point variables (or time-points) and $\mathcal{C}$ is a set of constraints, each having the form, $Y - X \leq \delta$, for some $X, Y \in \mathcal{T}$, and real number δ. Typically, the time-points in $\mathcal{T}$ represent starting or ending times of actions, or abstract coordination times. The constraints in $\mathcal{C}$ can accommodate release, deadline, duration and inter-action constraints. An STN is *consistent* if there exists a set of values for its time-points that together satisfy all of its constraints.

For any STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, there is a corresponding graph, $\mathcal{G}$, where the nodes in $\mathcal{G}$ correspond to the time-points in $\mathcal{T}$, and for each constraint, $Y - X \leq \delta$, in $\mathcal{C}$, there is an edge in $\mathcal{G}$ of the form, $X \xrightarrow{\delta} Y$. For convenience, this paper calls the constraints and edges in an STN *ordinary* constraints and edges.

The *all-pairs, shortest-paths* matrix for $\mathcal{G}$ is called the *distance matrix* for $\mathcal{S}$ (or $\mathcal{G}$) and is denoted by $\mathcal{D}$. Thus, for any X and Y in $\mathcal{T}$, $\mathcal{D}(X, Y)$ equals the length of the shortest path from X to Y in the graph $\mathcal{G}$. If $\mathcal{D}$ has nothing but zeros down its main diagonal, then $\mathcal{D}$ is said to be consistent.

Theorem 1 (Fundamental Theorem of STNs). *For any STN S, with graph G, and distance matrix D, the following are equivalent:*

- *S is consistent.*
- *G has no negative loops.*
- *D is consistent.*

1.2 STNs with Uncertainty

Some applications involve actions whose durations are uncontrollable, but nonetheless guaranteed to fall within known bounds. For example, when I turn on my laptop, I do not control how long it will take to load its operating system; however, I know that it will take anywhere from one to four minutes. A *Simple Temporal Network with Uncertainty (STNU)* augments an STN to include *contingent links* that represent this kind of uncontrollable-but-bounded temporal interval (Morris et al., 2001). A contingent link has the form, (A, x, y, C), where A and C are time-points and $0 < x < y < \infty$. A is called the *activation time-point; C* is called the *contingent time-point.* Intuitively, the duration of the interval from A to C is uncontrollable, but guaranteed to fall within the interval $[x, y]$. Typically, an agent controls the execution of the activation time-point A, but only *observes* the execution of the contingent time-point C in real time.[1]

Thus, an STNU is a triple, (T, C, L), where (T, C) is an STN, and L is a set of contingent links. N is used to denote the number of time-points in an STNU, K the number of contingent links.

The most important property of an STNU is whether it is *dynamically controllable* (DC)—that is, whether there exists a strategy for executing the non-contingent time-points that guarantees that all of the constraints in the network will be satisfied *no matter how the contingent durations turn out.* The strategy is *dynamic* in that its execution decisions are allowed to depend on and react to past observations, but not present or future observations. The formal semantics for dynamic controllability is quite complicated, but it need not be presented here because a more convenient—and equivalent—graphical characterization is available, as follows.

Graph for an STNU. Let $S = (T, C, L)$ be an STNU. The graph for S contains all of the edges from the STN, (T, C), as well as additional edges derived from the contingent links in L. In particular, for each contingent link $(A, x, y, C) \in L$, the graph contains the edges shown in Fig. 1. The ordinary

---

[1] Agents are not part of the formal semantics of STNUs; however, they are useful for expository purposes.
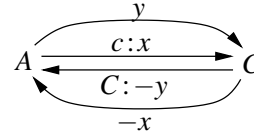

Figure 1: The edges associated with a contingent link.

edges, $A \xrightarrow{y} C$ and $C \xrightarrow{-x} A$, represent the ordinary constraints, $C - A \leq y$ and $A - C \leq -x$ (i.e., $C - A \in [x, y]$). The other two edges are *labeled* edges that represent uncontrollable *possibilities.* In particular, $A \xrightarrow{c:x} C$, which is called a *lower-case* (LC) edge, represents the possibility that the contingent duration might take on its minimum value, x; and $C \xrightarrow{C:-y} A$, which is called an *upper-case* (UC) edge, represents the possibility that the contingent duration might take on its maximum value, y.

Because the graph of an STNU contains ordinary, lower-case and upper-case edges, paths in an STNU graph can be quite complicated. However, as shall be seen, the so-called *semi-reducible* paths are particularly important. For expository convenience, the definition of a semi-reducible path is postponed; however, the *SR-distance matrix, D^*,* can be defined now as the all-pairs, shortest-*semi-reducible*-paths matrix for an STNU graph. Thus, for any time-points X and Y, $D^*(X, Y)$ equals the length of the shortest *semi-reducible* path from X to Y in the STNU graph.

Theorem 2 (Fundamental Theorem of STNUs). *For an STNU S, with graph G, and SR-distance matrix D^*, the following are equivalent:* [2]

- *S is dynamically controllable.*
- *G has no semi-reducible negative loops.*
- *D^* is consistent.*

1.3 DC-Checking Algorithms

In view of Theorem 2, the problem of determining whether an STNU is dynamically controllable can be answered by computing the SR-distance matrix D^*. If, during the process, a negative entry along the main diagonal is ever discovered—which would correspond to a semi-reducible negative loop—then the network cannot be dynamically controllable. Algorithms for determining whether an STNU is dynamically controllable are called *DC-checking algorithms.*

---

[2] Morris and Muscettola (2005) showed that an STNU is DC iff a certain matrix is consistent. Morris (2006) highlighted semi-reducible paths and showed that an STNU is DC iff its graph has no semi-reducible negative loops. Hunsberger (2010) showed that the matrix computed by Morris and Muscettola is equal to the SR-distance matrix.

Two polynomial-time DC-checking algorithms have been presented so far in the literature: the $O(N^5)$-time algorithm of Morris and Muscettola (2005), henceforth called the N^5 algorithm; and the $O(N^4)$-time algorithm of Morris (2006), henceforth called the N^4 algorithm.

The N^5 algorithm uses a set of rules to generate new edges in the graph, analogous to a new kind of constraint propagation that takes into account the different kinds of edges in an STNU. After at most $O(N^2)$ rounds of edge generation, the algorithm is guaranteed to have either computed the matrix $\mathcal{D}^*$ or determined that it is inconsistent. Since each round takes $O(N^3)$ time, the overall complexity is $O(N^5)$.

The N^4 algorithm uses the same edge-generation rules but, as will be seen, restricts their application to "reducing away" LC edges. This restricted form of edge-generation is sufficient to compute the matrix $\mathcal{D}^*$ or determine that it is inconsistent. Based on an analysis of the structure of semi-reducible negative loops, the N^4 algorithm requires only $K \leq N$ rounds of edge-generation. Since each round can be done in $O(N^3)$ time, its overall time-complexity is $O(N^4)$.

1.3.1 Edge-Generation Rules

Intuitively, the ordinary constraints in an STNU are constraints that the agent in charge of executing time-points wants to satisfy. In contrast, the lower-case and upper-case edges represent uncontrollable *possibilities* that could potentially threaten the satisfaction of the ordinary constraints. Typically, to eliminate such threats, the agent must satisfy additional constraints—or, in graphical terms, add new edges to the graph. Toward that end, Morris and Muscettola (2005) presented the five edge-generation rules in Table 1, where pre-existing edges are denoted by solid arrows and newly generated edges are denoted by dashed arrows. The first four rules each take two pre-existing edges as input and generates a single edge as output. The Label-Removal rule takes only one edge as input. Incidentally, applicability conditions of the form, $Y \not\equiv Z$, should be construed as stipulating that Y and Z must be distinct time-point variables, not as constraints on the *values* of those variables.

Note that the rules only generate new ordinary or upper-case edges, never new lower-case edges. The generated ordinary edges represent additional constraints that must be satisfied to avoid threatening the satisfaction of the original constraints. The generated upper-case edges represent additional *conditional* constraints that the agent must satisfy. In particular, a generated UC edge of the form, $Y \xrightarrow{\;C:-w\;} A$, represents a conditional constraint that can be glossed
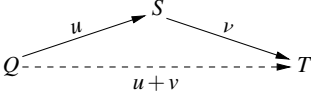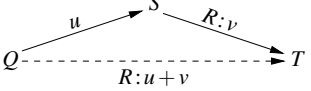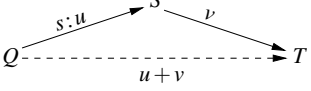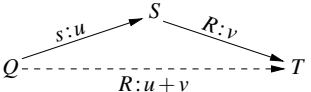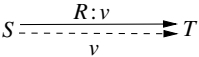

Table 1: Edge-generation rules.

Figure 2: Transforming a path $\mathcal{P}$ into the path $\mathcal{P}'$.

as: "As long as the contingent duration $C - A$ might take on its maximum value, then $A - Y \leq -w$ (i.e., $Y \geq A + w$) must be satisfied".

1.3.2 Path Transformations

Morris (2006) showed that the process of edge generation can also be viewed as one of *path transformation* or *path reduction*. For example, as illustrated in Fig. 2, suppose a path $\mathcal{P}$ contains two adjacent edges, e_1 and e_2, to which one of the first four edge-gene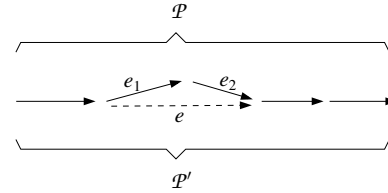ration rules can be applied to generate a new edge e. Let $\mathcal{P}'$ be the path obtained from $\mathcal{P}$ by replacing e_1 and e_2 by the new edge e. We say that $\mathcal{P}$ has been *transformed into* (or *reduced* to) $\mathcal{P}'$. Similarly, if $\mathcal{P}$ contains a UC edge E to which the Label-Removal rule can be applied to generate a new ordinary edge E^o, then $\mathcal{P}$ can be transformed by replacing E by E^o. Finally, any *sequence* of zero or more such transfor-
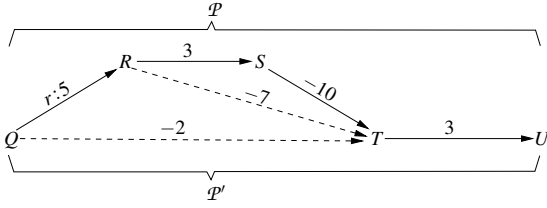
Figure 3: A two-step transformation of a path $\mathcal{P}$ into $\mathcal{P}'$.

Figure 4: Reducing away a lower-case edge, e.

mations also counts as a path transformation. Fig. 3 illustrates the transformation of a path $\mathcal{P}$ using the No-Case rule followed by the Lower-Case rule.

An important property of path transformations is that they preserve *unlabeled length* (i.e., the length of the path ignoring any alphabetic labels on its edges). This follows directly from the fact that each edge-generation rule preserves unlabeled length.

Morris (2006) introduced *semi-reducible* paths, which play a central role in the determination of dynamic controllability. For convenience, we present the definition in terms of OU-edges and OU-paths.

Definition 1 (OU-edge, OU-path). An *OU-edge* is an edge that is either ordinary or upper-case. An *OU-path* is a path consisting solely of OU-edges.

Definition 2 (Semi-reducible path; SRN loop). A path in an STNU graph is called *semi-reducible* if it can be transformed into an OU-path. A semi-reducible loop with negative unlabeled length is called an *SRN loop*.

Note that the path, $\mathcal{P}$, in Fig. 3 is semi-reducible since it can be transformed into the OU-path, $\mathcal{P}'$.

1.3.3 The N^4 DC-Checking Algorithm

The N^4 algorithm takes a two-step approach to determining whether an STNU has any SRN loops. In Step 1, it generates the OU-edges that could arise from the transformation of semi-reducible paths into OU-paths. The dashed edges in Fig. 3 are examples of such edges. In Step 2, it gathers the OU-edges from Step 1—minus any alphabetic labels—into an STN, $\mathcal{S}^\dagger$. It then computes the corresponding distance matrix, $\mathcal{D}^\dagger$, which turns out to equal the SR-distance matrix, $\mathcal{D}^*$, for the original STNU (i.e., the all-pairs, shortest-*semi-reducible*-paths matrix).

To illustrate Step 1, consider a semi-reducible path, $\mathcal{P}$, that consists of original STNU edges, including at least one lower-case edge e, as shown in Fig. 4. Since $\mathcal{P}$ is semi-reducible, there must be some sequence of reductions by which $\mathcal{P}$ is transformed into an OU-path. Thus, sometime during that transformation, the Lower-Case or Cross-Case rule must be applied to e and some other edge e' to yield a new
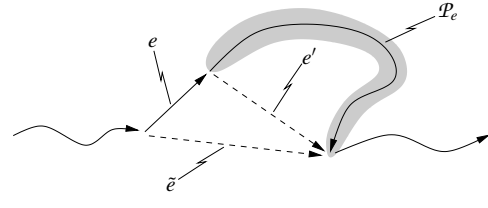
OU-edge $\tilde{e}$, effectively removing e from the path. We say that e has been "reduced away". To enable this, the original path $\mathcal{P}$ must have a sub-path, $\mathcal{P}_e$, immediately following e, such that $\mathcal{P}_e$ reduces to the edge e', as shown in Fig. 4.[3] The concatenation of the LC edge e with the sub-path $\mathcal{P}_e$ is called a *lower-case reducing sub-path* (LCR sub-path); edges such as $\tilde{e}$ that are generated by transforming an LCR sub-path into a single edge, are called *core edges* (Hunsberger, 2010).

In view of the above, it follows that every occurrence of a lower-case edge, e, in any semi-reducible path, $\mathcal{P}$, must belong to an LCR sub-path in $\mathcal{P}$. Stated differently, the edges in any semi-reducible path, $\mathcal{P}$, that do *not* belong to an LCR sub-path must be OU-edges from the original STNU. Thus, Step 1 of the N^4 algorithm searches for LCR sub-paths and the core edges that they generate. Crucially, this search does not require exhaustively applying the edge-generation rules from Table 1. Instead, as will be seen, the search can be limited to so-called *extension sub-paths,* which have an important nesting property. At the end of Step 1, the algorithm has a set, $\mathcal{E}$, of OU-edges.

For Step 2, note that there is a one-to-one correspondence between shortest semi-reducible paths in the original STNU and shortest paths consisting of edges in $\mathcal{E}$. In particular, if $\mathcal{P}$ is a shortest semi-reducible path, then it can be transformed into a path, $\mathcal{P}'$, whose edges are in $\mathcal{E}$; and since path transformations preserve unlabeled length, $|\mathcal{P}| = |\mathcal{P}'|$. Similarly, if $\mathcal{P}'$ is a shortest path with edges in $\mathcal{E}$, then, by "unwinding" the transformations that generated the edges in $\mathcal{E}$, $\mathcal{P}'$ can be "un-transformed" into a semi-reducible path $\mathcal{P}$, again, such that $|\mathcal{P}'| = |\mathcal{P}|$.

Next, since alphabetic labels are irrelevant to the computation of unlabeled lengths, let $\mathcal{E}^\dagger$ be the set of ordinary edges obtained by stripping any alphabetic labels from the edges in $\mathcal{E}$; let $\mathcal{S}^\dagger$ be the corresponding STN; and let $\mathcal{D}^\dagger$ be the corresponding distance matrix. Then $\mathcal{D}^\dagger$ is equal to the all-pairs, shortest-paths matrix for paths with edges in $\mathcal{E}$, and hence $\mathcal{D}^\dagger = \mathcal{D}^*$. Thus, the N^4 algorithm concludes that the original STNU is DC iff $\mathcal{D}^\dagger$ is consistent.

---

[3] In Fig. 3, the LC edge from Q to R is reduced away by the path from R to S to T, yielding an edge from Q to T.

2 Modifying Morris' Analysis

To simplify his mathematical analysis, Morris (2006) introduces two kinds of instantaneous reactivity into the semantics of dynamic controllability. First, he allows contingent links of the form, $(A, 0, y, C)$, in which the lower bound on the contingent duration is zero. This effectively allows scenarios in which it is uncertain whether the temporal interval between a cause and its effect will be instantaneous. Second, he allows an agent to react instantaneously to an observation of a contingent execution. Although these sorts of instantaneous reactions may be applicable to some domains, this author prefers to stick with the more realistic assumptions of the original semantics—and the edge-generation rules in Table 1—in which both the lower bounds of contingent durations and agent reaction times must be positive. The rest of this section shows how Morris' concepts and techniques must be modified to conform to the original semantics of dynamic controllability.

- For convenience, proofs for the results in the rest of the paper are sketched in the Appendix.

Given his assumptions, Morris changed the conditions for the Lower-Case rule to $v < 0$ (i.e., he eliminated the case, $v = 0$). The reason is that when $v = 0$, the edge, $S \xrightarrow{v} T$, represents the constraint, $T - S \leq 0$ (i.e., $T \leq S$), which expresses that T must execute no later than the contingent time-point S. If able to react instantaneously, an agent need only wait for S to execute and then instantaneously execute T. Thus, no additional constraint is required to guard against S executing early. If unable to react instantaneously, then the new edge from Q to T is needed. Similar remarks apply to the Cross-Case rule.

Extension Sub-Paths. Let e be some LC edge in a path, $\mathcal{P}$; and let $e_1, e_2, \ldots$ be the sequence of edges immediately following e in $\mathcal{P}$. If e can be reduced away in $\mathcal{P}$, then it may be that there are many values of $m \geq 1$ for which the sub-path, $e_1, e_2, \ldots, e_m$, could be used to reduce away e. For example, the LC edge from Q to R in Fig. 3 can be reduced away not only by the two-edge sub-path from R to T, as shown in the figure, but also by the three-edge sub-path from R to U. In such cases, the *extension sub-path,* defined below, will turn out to be the sub-path that can reduce away e for which the value of m is the smallest.

Definition 3 (Extension sub-path; moat edge). Let e be an LC edge in a path $\mathcal{P}$. Let $e_1, e_2, \ldots$ be the sequence of edges that immediately follow e in $\mathcal{P}$. For each $i \geq 1$, let $\mathcal{P}_e^i$ be the sub-path of $\mathcal{P}$ consisting of the edges, $e_1, \ldots, e_i$. If it exists, let m be the smallest integer such that either:
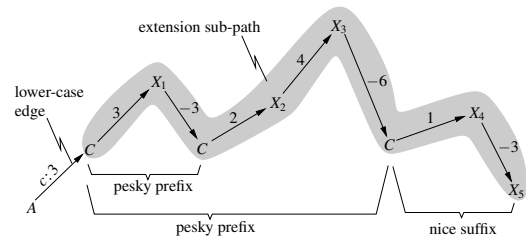

Figure 5: An ESP with two pesky prefixes.

(1) $|\mathcal{P}_e^m| < 0$; or
(2) $|\mathcal{P}_e^m| = 0$ and $\mathcal{P}_e^m$ is *not* a loop.[4]

Then the *extension sub-path* (ESP) for e in $\mathcal{P}$, notated $\mathcal{P}_e$, is the sub-path $\mathcal{P}_e^m$; and its last edge, e_m, is called the *moat edge* for e in $\mathcal{P}$. If no such m exists, then e has no ESP or moat edge in $\mathcal{P}$.

For the LC edge from Q to R in Fig. 3, the extension sub-path is the two-edge path labeled $\mathcal{P}_e$; and the moat edge is the edge from S to T.

Structure of ESPs. Given the setup in Defn. 3, m is the smallest value for which $|\mathcal{P}_e^m| < 0$ or $\mathcal{P}_e^m$ is a zero-length non-loop. Conversely, for any $i < m$, either $|\mathcal{P}_e^i| > 0$ or $\mathcal{P}_e^i$ is a zero-length loop. In turn, this implies that any ESP must consist of *zero or more* loops of length zero, followed by a (non-empty) sub-path that does not have any prefixes that are zero-length loops. These observations motivate the following.

Definition 4 (Pesky prefix; nice path). A *pesky prefix* of $\mathcal{P}$ is a non-empty prefix of $\mathcal{P}$ that is a loop of length 0. A *nice* path is one having no pesky prefixes.

In general, an ESP may have zero or more pesky prefixes, followed by a non-empty nice path.[5] Fig. 5 shows an ESP with two pesky prefixes, one nested inside the other, followed by a non-empty nice suffix.

Nesting Property for ESPs. The following lemma confirms that ESPs as defined in Defn. 3 have the nesting property highlighted by Morris (2006).

Lemma 1 (Nesting Property for ESPs). *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two ESPs within the same path $\mathcal{P}$. Then $\mathcal{P}_1$ and $\mathcal{P}_2$ are either disjoint (i.e., share no edges) or one is nested inside (i.e., is a sub-path of) the other.*

Breaches and Usable/Unusable Moat Edges. Suppose that $\mathcal{P}$ is a path that contains an occurrence of a lower-case edge, e, that derives from a contingent link, (A, x, y, C). Thus, e has the form, $A \xrightarrow{c:x} C$.

---

[4]For Morris (2006), case (2) is not needed because he eliminated the case, $v = 0$, from the applicability conditions for the Lower-Case and Cross-Case rules.

[5]Unlike Morris (2006), for whom every ESP has negative length, this paper must carefully distinguish pesky prefixes from ESPs of length zero.
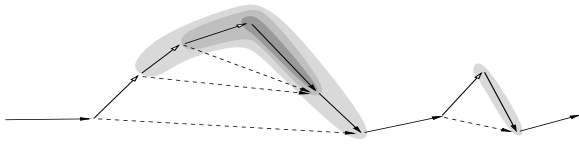
Figure 6: A semi-reducible path with nested ESPs.

Figure 7: An indivisible SRN loop, $\mathcal{P}$.

Suppose further that e has an extension sub-path, $\mathcal{P}_e$, in $\mathcal{P}$. The existence of an ESP for e in $\mathcal{P}$ turns out to be a necessary, but insufficient condition for reducing away e in $\mathcal{P}$. For example, using Fig. 4 as a reference, if the edge, e', into which $\mathcal{P}_e$ is transformed, happens to be an upper-case edge with alphabetic label C (i.e., that matches the lower-case label on e), then the Cross-Case rule cannot be applied to e and e', blocking the reducing away of e.[6] Such moat edges are called *unusable*.[7] The following definitions specify the characteristics of usable/unusable moat edges.

Definition 5 (Breach; usable/unusable moat edge). Let e be a lower-case edge corresponding to a contingent link, (A, x, y, C); let $\mathcal{P}_e$ be the ESP for e in some path $\mathcal{P}$; and let e_m be the corresponding moat edge. Any occurrence in $\mathcal{P}_e$ of an upper-case edge labeled by C is called a *breach*. If $\mathcal{P}_e$ has no breaches, then it is called *breach-free*. If e_m is a breach and $|\mathcal{P}_e| < -x$, then e_m is said to be *unusable;* otherwise, it is *usable*.[8]

Theorem 3, below, shows the crucial role of usable moat edges for semi-reducible paths (Morris 2006).

Theorem 3. *A path $\mathcal{P}$ is semi-reducible if and only if each of its lower-case edges has a usable moat edge.*

Since a pesky prefix, by definition, has length zero, extracting a pesky prefix from an extension sub-path cannot affect its length. In addition, since a pesky prefix cannot constitute the entirety of an extension sub-path, extracting a pesky prefix cannot affect the moat edge. Therefore, the usability of a moat edge cannot be affected by extracting a pesky prefix from an ESP and, hence, the semi-reducibility of a path cannot be affected by extracting pesky prefixes.

Corollary 1. *Let $\mathcal{P}$ be any path. Let $\mathcal{P}'$ be the path obtained from $\mathcal{P}$ by extracting all pesky prefixes from any extension sub-paths within $\mathcal{P}$. Then $\mathcal{P}$ is semireducible if and only if $\mathcal{P}'$ is semi-reducible.*

Given this result, the rest of this paper presumes that all pesky prefixes are extracted from any path without affecting its semi-reducibility.
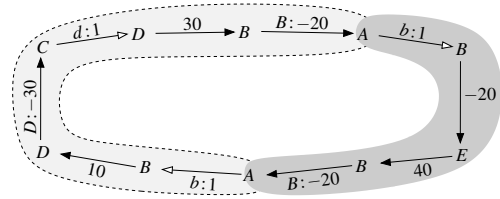
Corollary 2. *Any semi-reducible path, $\mathcal{P}$, can be transformed into an OU-path using a sequence of re-*

ductions whereby each LC edge e in $\mathcal{P}$ is reduced away by its corrresponding extension sub-path $\mathcal{P}_e$.

Fig. 6 illustrates a semi-reducible path with nested extension sub-paths. In the figure, lower-case edges are shown with a distinctive arrow type, ESPs are shaded, and the core edges are dashed.

Morris (2006) proved that an STNU with K contingent links has an SRN loop if and only if it has a breach-free SRN loop in which extension sub-paths are nested to a depth of at most K. Thus, his N^4 DC-checking algorithm performs K rounds of searching for breach-free extension sub-paths that could be used to reduce away lower-case edges, each round effectively increasing the nesting depth of extension sub-paths. The core edges generated in this way are then collected—minus any upper-case labels—into an STN, $\mathcal{S}^\dagger$, as previously described, to compute the distance matrix, $\mathcal{D}^\dagger$, which equals the all-pairs, shortest-*semi-reducible*-paths matrix for the original STNU.

3 INDIVISIBLE SRN LOOPS

This section introduces a new approach to analyzing the structure of semi-reducible negative loops. The key feature of the approach is its focus on the number of occurrences of lower-case edges in what it calls *indivisible* SRN loops (or iSRN loops). As will be seen, for the purposes of DC checking, it suffices to restrict attention to iSRN loops. However, the main result of this section is that the number of occurrences of LC edges in any iSRN loop in any STNU having K contingent links is at most $2^K - 1$. Section 4 shows how to construct STNUs that have iSRN loops—called *magic loops*—that attain this upper bound. Section 5 exploits the $2^K - 1$ bound to speed up the DC checking for some STNUs.

The measure of complexity considered below is the number of occurrences of LC edges in a path.

Definition 6. For any path, $\mathcal{P}$, the number of occurrences of lower-case edges in $\mathcal{P}$ is denoted by $\#\mathcal{P}$.

Suppose that $\mathcal{P}$ is an SRN loop and Q is a subloop of $\mathcal{P}$ that also happens to be an SRN loop (i.e., Q

---

[6]Recall the condition, $R \not\equiv S$, for the Cross-Case rule.

[7]It could also be said that $\mathcal{P}_e$ is unusable.

[8]Morris (2006) converts STNUs into a normal form in which the value of x is invariably zero.

is an SRN sub-loop of $\mathcal{P}$). Since every LC edge in Q also belongs to $\mathcal{P}$, it follows that $\#Q \leq \#\mathcal{P}$. However, if $\mathcal{P}$ is an *indivisible* SRN loop, then $\#Q$ must equal $\#\mathcal{P}$. That is, no SRN sub-loop of an iSRN loop $\mathcal{P}$ can have *fewer* occurrences of LC edges than $\mathcal{P}$.

Definition 7 (iSRN loop). Let $\mathcal{P}$ be an SRN loop. $\mathcal{P}$ is called an *indivisible* SRN loop (or iSRN loop) if $\#Q = \#\mathcal{P}$ for every SRN sub-loop Q of $\mathcal{P}$.

Fig. 7 shows an example of an SRN loop, $\mathcal{P}$, that has no SRN sub-loops and, thus, is indivisible. $\mathcal{P}$ contains three occurrences of LC edges (shown with a distinctive arrow style); that is, $\#\mathcal{P} = 3$. $\mathcal{P}$ is semi-reducible because each LC edge has a corresponding breach-free extension sub-path that can be used to reduce it away. In addition, $|\mathcal{P}| = -7 < 0$. Finally, although $\mathcal{P}$ has many sub-loops, two of which are shaded in the figure, none of them are SRN sub-loops. For example, the lefthand shaded sub-loop is not semi-reducible and the righthand shaded sub-loop is non-negative. Thus, $\mathcal{P}$ is an iSRN loop.

Lemma 2, below, shows that for DC checking, it suffices to restrict attention to iSRN loops. The iSRN loop, $\mathcal{P}'$, is obtained by recursively extracting SRN sub-loops until, eventually, an iSRN loop is found.

Lemma 2. *If an STNU $\mathcal{S}$ has an SRN loop $\mathcal{P}$, then $\mathcal{S}$ also has an iSRN loop $\mathcal{P}'$. Furthermore, $\mathcal{P}'$ can be chosen such that $\#\mathcal{P}' \leq \#\mathcal{P}$.*

The search for an upper bound on the number of occurrences of LC edges in any iSRN loop begins by focusing on *root-level* LCR sub-paths (i.e., LCR sub-paths that are not contained within any other). This notion can be defined since, by Lemma 1, ESPs in any semi-reducible path must be disjoint or nested.

Definition 8 (Root-level). Let e be an occurrence of an LC edge in a semi-reducible path $\mathcal{P}$; and let $\mathcal{P}_e$ be the extension sub-path for e in $\mathcal{P}$. If $\mathcal{P}_e$ is not contained within any other ESP in $\mathcal{P}$, then $\mathcal{P}_e$ is called a *root-level* ESP in $\mathcal{P}$; e is called a *root-level* LC edge in $\mathcal{P}$; and the LCR sub-path formed by concatenating e and $\mathcal{P}_e$ is called a *root-level* LCR sub-path.

Theorem 4 bounds the number of root-level LCR sub-paths in any iSRN loop.

Theorem 4. *An iSRN loop in an STNU with K contingent links has at most K root-level LCR sub-paths.*

Theorem 5, below, bounds the depth of nesting of LCR sub-paths (or, equivalently, ESPs) in an iSRN loop. It extends Morris' result that if an STNU with K contingent links has an SRN loop, then it has a breach-free SRN loop whose extension sub-paths are nested to a depth of at most K.

Theorem 5. *Let $\mathcal{P}$ be an iSRN loop in an STNU having K contingent links. Then $\mathcal{P}$ is breach-free and has LCR sub-paths nested to a depth of at most K.*

Although Theorem 5 bounds the nesting depth of LCR sub-paths in an iSRN loop, it does not limit the number of LC edges within any root-level LCR sub-path. Theorem 6, below, shows that in any non-trivial iSRN loop there must be an LC edge that occurs exactly once, and that that occurrence must be at the root level.

Theorem 6. *If $\mathcal{P}$ is an iSRN loop that contains at least one lower-case edge, then $\mathcal{P}$ must have a root-level LC edge that occurs exactly once in $\mathcal{P}$.*

This result provides the key for the inductive proof of Theorem 7, below, the main result of this section.

Theorem 7. *If $\mathcal{P}$ is an iSRN loop in an STNU with K contingent links, then $\#\mathcal{P} \leq 2^K - 1$.*

Finally, Theorem 8 shows that the *ordinary* edges associated with contingent links (cf. Fig. 1) can be ignored for the purposes of DC checking.

Theorem 8. *Any STNU having an SRN loop has an iSRN loop that contains none of the ordinary edges associated with contingent links.*
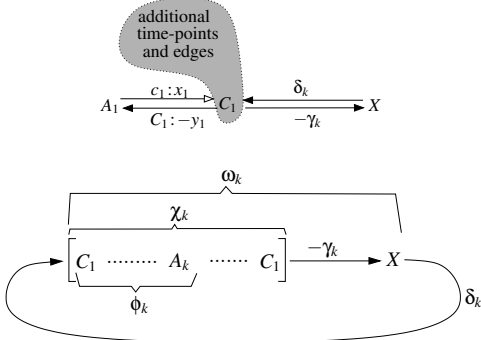
Although this does not affect the worst-case complexity of DC checking, it has the potential to limit the branching factor of edge generation in practice.

4 MAGIC LOOPS

Section 3 showed that the number of LC edges in any iSRN loop is at most $2^K - 1$. This section defines a *magic loop* as any iSRN loop that has exactly $2^K - 1$ occurrences of LC edges. It then presents an algorithm for constructing such loops, thereby proving that the $2^K - 1$ bound is tight. Interestingly, the STNUs used to generate these magic loops have only $2K + 1$ time-points (two time-points for each contingent link, plus one extra time-point) and $4K$ edges.

Definition 9 (Magic Loop). A *magic loop of order K* is any iSRN loop that (1) belongs to an STNU having K contingent links; and (2) contains exactly $2^K - 1$ occurrences of LC edges

The algorithm for constructing magic loops is recursive. For each $K \geq 1$, it defines an STNU, $\mathcal{S}_K$, that contains a magic loop, $\mathcal{M}_K$, of order K. The STNUs and magic loops employ edges whose lengths are specified by numerical parameters, such as $x_i, y_i, \alpha_i, \beta_i, \gamma_i$, and δ_i, where $1 \leq i \leq K$. All of these parameters have positive integer values; thus, any negative values are specified with explicit negative signs, as in: $-y_i, -\alpha_i$ or $-\gamma_i$. Each magic loop, $\mathcal{M}_K$, also has several sub-paths, including ϕ_i, χ_i and ω_i. These sub-paths have important properties that are exploited in the related proofs. It shall turn out
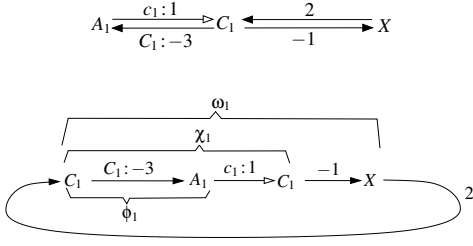
Figure 8: The STNU $\mathcal{S}_1$ (top) and magic loop $\mathcal{M}_1$ (bottom).

Figure 9: The generic form of $\mathcal{S}_k$ (top) and $\mathcal{M}_k$ (bottom).

Figure 10: Building the STNU, $\mathcal{S}_*$, from $\mathcal{S}_k$.

Figure 12: STNU $\mathcal{S}_2$ (top) and magic loop $\mathcal{M}_2$ (bottom).

that all of the parameters are positive; however, the lengths, $|\phi_i|, |\chi_i|$ and $|\omega_i|$ are invariably negative.

$\Rightarrow$ For convenience, the rest of this section uses k instead of K, and $*$ instead of $k+1$. Thus, for example, $\mathcal{S}_*$ is shorthand for $\mathcal{S}_{K+1}$.
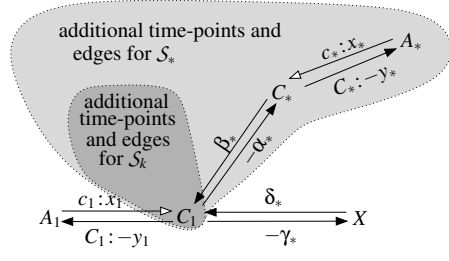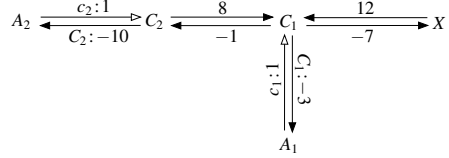
For the base case, the STNU $\mathcal{S}_1$ and its magic loop $\mathcal{M}_1$ are shown in Fig. 8. $\mathcal{M}_1$ contains two sub-loops, neither of which is an SRN loop; thus, it is an iSRN loop. Also, $\mathcal{M}_1$ contains $2^1 - 1 = 1$ occurrence of an LC edge; thus, it is a magic loop of order 1.

For the recursive case, we assume that we have $\mathcal{S}_k$, an STNU with k contingent links that has the form shown at the top of Fig. 9, and $\mathcal{M}_k$, a magic loop of order k having the form shown at the bottom of Fig. 9. Note that $\mathcal{S}_1$ and $\mathcal{M}_1$ have the desired forms.

In general, the values of $\gamma_k, |\phi_k|$ and $|\chi_k|$ suffice to generate the values of the parameters, $\alpha_*, \beta_*, \gamma_*, \delta_*, x_*$ and y_*, which are determined sequentially, as shown in Rules 1–6 of Table 2.[9] Once these values are in

---

[9]Recall that the asterisk is used as a shorthand for $k+1$.

$$
\begin{array}{ll}
(1) & \alpha_* = \gamma_k \\
(2) & \beta_* = 1 - 2|\phi_k| + \gamma_k \\
(3) & \gamma_* = 2 - 2|\phi_k| + |\chi_k| + \gamma_k \\
(4) & \delta_* = 2 - 3|\phi_k| + \gamma_k \\
(5) & x_* = 1 \\
(6) & y_* = 3 - 3|\phi_k| + |\chi_k|
\end{array}
$$

Table 2: Rules for generating parameters for the case $k+1$.

hand, the STNU, $\mathcal{S}_*$, is built out of $\mathcal{S}_k$ as shown in Fig. 10; and the magic loop, $\mathcal{M}_*$, is created with the structure shown in Fig. 11.

Notice, too, that $\mathcal{M}_*$ introduces a single, new lower-case edge associated with a contingent link, (A_*, x_*, y_*, C_*). Since each χ_k sub-path has $2^k - 1$ occurrences of LC edges, the total number of occurrences of LC edges in $\mathcal{M}_*$ is $1 + 2(2^k - 1) = 2^{k+1} - 1$, as desired. Fig. 12 shows the STNU $\mathcal{S}_2$ and magic loop $\mathcal{M}_2$ generated using these parameters; the LCR sub-paths are shaded for convenience.

Although the structure of magic loops of higher orders is extremely convoluted, the structure of the corresponding STNU graphs is much simpler. For example, the STNU, $\mathcal{S}_5$, is shown in Fig. 13.

Finally, Theorem 9, below, shows that for each $k \geq 1$, the loop, $\mathcal{M}_k$, is indeed a magic loop; and Theorem 10 shows that for each $k \geq 1$, the only iSRN loops in $\mathcal{S}_k$ are necessarily magic loops; thus, there are no iSRN loops in $\mathcal{S}_k$ having fewer than $2^k - 1$ occurrences of lower-case edges. Taken together, these theorems show that magic loops are not only worst-case scenarios in terms of the number of occurrences of LC edges in an iSRN, but also that there are STNUs for which this worst-case scenario is the only case.

Theorem 9. *For each $k \geq 1$, the loop, $\mathcal{M}_k$, is a magic loop of order k (i.e., an iSRN loop having exactly $2^k - 1$ occurrences of lower-case edges).*

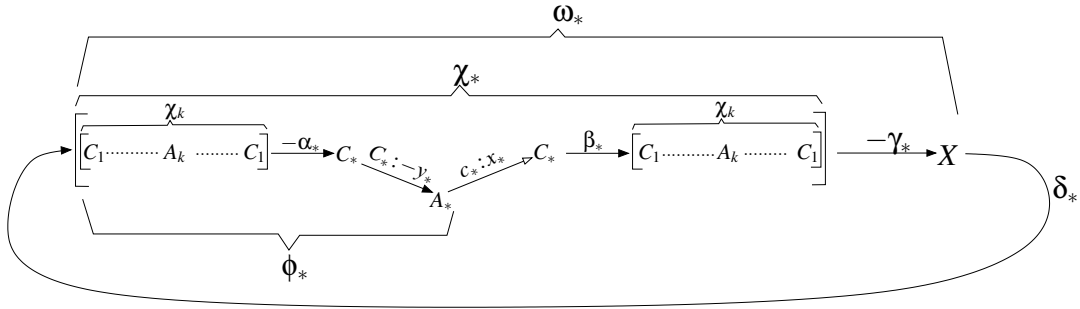**Theorem 10.** *Let $\mathcal{S}_k$ be the STNU as described in this*

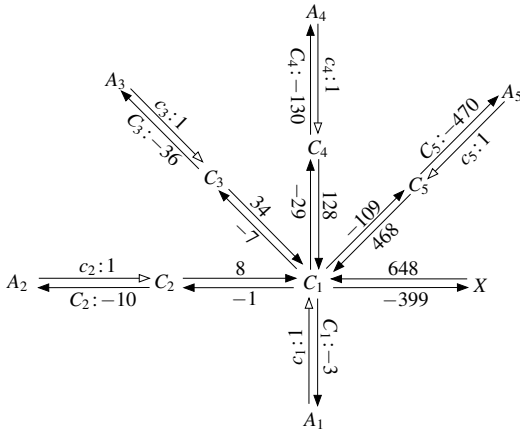Figure 11: The structure of $\mathcal{M}_*$, a magic loop of order $k+1$.

Figure 13: The STNU $\mathcal{S}_5$ that contains the magic loop $\mathcal{M}_5$.

section for some $k \geq 1$. Every SRN loop in $\mathcal{S}_k$ has at least $2^k - 1$ occurrences of LC edges.

5 SPEEDING UP DC CHECKING

This section presents a recursive $O(N^3)$-time pre-processing algorithm that exploits the $2^K - 1$ bound on the number of occurrences of LC edges in iSRN loops. For certain networks, this pre-processing algorithm decreases the computation time for the N^4 DC-checking algorithm from $O(N^4)$ to $O(N^3)$.

Let $\mathcal{S}$ be an STNU having K contingent links. The pre-processing algorithm computes, for each contingent time-point, C_j, an upper bound on the number of distinct contingent time-points that can co-occur in any iSRN loop in $\mathcal{S}$ that contains C_j. The largest of these upper bounds then serves as an upper bound, UB, on the number of distinct contingent time-points—and hence the number of distinct LC edges—that can co-occur in any single iSRN loop in $\mathcal{S}$. Since any iSRN loop having at most UB distinct lower-case edges can be viewed as an iSRN loop in an STNU having exactly UB contingent links, such a loop can have extension sub-paths nested to a depth of at most

UB (cf. Theorem 5). Thus, UB also provides an upper bound on the number of rounds needed for the N^4 algorithm to check the dynamic controllability of $\mathcal{S}$.

In cases where $UB < K$, the pre-processing algorithm can provide significant savings. Indeed, for some STNUs, $UB = 1$, implying the need for only one $O(N^3)$-time round of the N^4 algorithm, even though the unaware N^4 algorithm might still perform K rounds at a cost of $O(N^4)$. At the other extreme, for some STNUs, $UB = K$, in which case, the pre-processing algorithm provides no benefit. However, since the pre-processing algorithm runs in $O(N^3)$ time, it does not introduce a significant overhead for the N^4 algorithm, whose first step is an $O(N^3)$-time computation of a distance matrix.

In more detail. Given an STNU, $\mathcal{S}$, with K contingent links, the algorithm begins by computing:

- $2^K - 1$, the maximum number of occurrences of LC edges in any iSRN loop in $\mathcal{S}$;
- Δ, the maximum value of $y - x$ among all of the contingent links, (A, x, y, C), in $\mathcal{S}$; and
- $\mathcal{D}^\diamond$, the all-pairs, shortest-path matrix for the OU-paths in $\mathcal{S}$, which can be computed in $O(N^3)$ time.

Next, for each pair of *distinct* contingent time-points, C_i and C_j, it computes the value, LB_{ij}:

$$LB_{ij} = \mathcal{D}^\diamond(C_i, C_j) + \mathcal{D}^\diamond(C_j, C_i) - (2^K - 1)\Delta.$$

As will be shown, if $\mathcal{P}$ is any iSRN loop in $\mathcal{S}$ that contains both C_i and C_j, then $|\mathcal{P}| \geq LB_{ij}$ (i.e., LB_{ij} is a *Lower Bound* for the lengths of iSRN loops that contain both C_i and C_j). Thus, if $LB_{ij} \geq 0$, it follows that C_i and C_j cannot co-occur in any iSRN loop in $\mathcal{S}$. But in that case, any iSRN loop—if such exists—can have at most $K - 1$ *distinct* LC edges and, thus, no more than $2^{(K-1)} - 1$ *occurrences* of LC edges.

If the upper bound on the number of occurrences of LC edges in iSRN loops in $\mathcal{S}$ can be lowered in this way, the algorithm recursively seeks to identify additional combinations of contingent time-points that cannot co-occur within any iSRN loop. It terminates when no further combinations can be found.
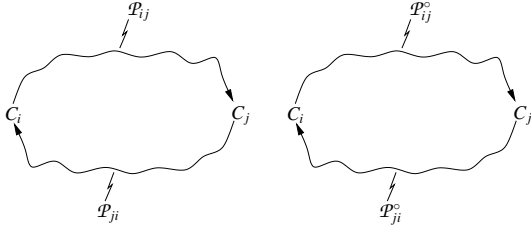
Figure 14: The iSRN loop, $\mathcal{P}$, and its OU-cousin, $\mathcal{P}^\circ$.

Consider the scenario in Fig. 14, where the left-hand loop is an iSRN loop, $\mathcal{P}$, that contains a pair of *distinct* contingent time-points, C_i and C_j. Note that the sub-path from C_i to C_j is called $\mathcal{P}_{ij}$, and the sub-path from C_j to C_i is called $\mathcal{P}_{ji}$. Next, define the *ordinary cousin* of an LC edge, $A \xrightarrow{\;x\;} C$, to be the corresponding ordinary edge, $A \xrightarrow{\;y\;} C$, for the contingent link (A, x, y, C) (cf. Fig. 1). The righthand loop, $\mathcal{P}^\circ$, in Fig. 14 is the same as $\mathcal{P}$, except that any occurrences of LC edges have been replaced by their *ordinary cousins*. Since $\mathcal{P}^\circ$ may yet contain upper-case edges, we call it the *OU-cousin* of $\mathcal{P}$. Notice that $\mathcal{P}^\circ$ is the concatenation of the OU-cousins of $\mathcal{P}_{ij}$ and $\mathcal{P}_{ji}$. Furthermore, since $\mathcal{P}_{ij}^\circ$ and $\mathcal{P}_{ji}^\circ$ are OU-paths, it follows that their lengths are bounded below by the corresponding OU-distance-matrix entries, whence:

$$\mathcal{D}^\diamond(C_i, C_j) + \mathcal{D}^\diamond(C_j, C_i) \;\leq\; |\mathcal{P}_{ij}^\circ| + |\mathcal{P}_{ji}^\circ| \;=\; |\mathcal{P}^\circ| \quad (1)$$

Now, by Theorem 7, since $\mathcal{P}$ is an iSRN loop, $\#\mathcal{P} \leq 2^K - 1$. Thus, the difference in the lengths of $\mathcal{P}$ and $\mathcal{P}^\circ$ is bounded as follows:

$$\Delta_{\mathcal{P}} \;=\; |\mathcal{P}^\circ| - |\mathcal{P}| \;\leq\; \#\mathcal{P}(2^K - 1)\Delta \;\leq\; (2^K - 1)\Delta \quad (2)$$

where Δ is the maximum value of $y - x$ over all the contingent links in the STNU. Combining the inequalities (1) and (2) then yields:

$$\begin{aligned} |\mathcal{P}| \;&\geq\; |\mathcal{P}^\circ| - (2^K - 1)\Delta \\ &\geq\; \mathcal{D}^\diamond(C_i, C_j) + \mathcal{D}^\diamond(C_j, C_i) - (2^K - 1)\Delta \end{aligned}$$

Since this inequality must hold whenever $\mathcal{P}$ is an iSRN loop in which the distinct contingent time-points, C_i and C_j, both occur, it follows that if

$$\mathcal{D}^\diamond(C_i, C_j) + \mathcal{D}^\diamond(C_j, C_i) - (2^K - 1)\Delta \geq 0$$

then there cannot be any such loop. (Recall that $|\mathcal{P}|$ must be negative if $\mathcal{P}$ is an iSRN loop.)

Next, if for each pair of contingent time-points, C_i and C_j, we define $\mathcal{F}(i, j) = \mathcal{D}^\diamond(C_i, C_j) + \mathcal{D}^\diamond(C_j, C_i)$, then the preceding rule, which is the main rule used by the pre-processing algorithm, can be re-stated as:

- If C_i and C_j are distinct contingent time-points such that $\mathcal{F}(i, j) \geq (2^K - 1)\Delta$, then C_i and C_j cannot both occur in the same iSRN loop.

Pseudo-code for the pre-processing algorithm is given in Table 3. For each contingent time-point, C_i, it defines the following variables:

Given: An STNU $\mathcal{S}$ with K contingent links.

0. Initialization:
   - Let Δ and $\mathcal{F}$ be as defined in the text.
   - For each $i \in \{1, 2, \ldots, K\}$,
     ○ $\text{ctr}_i := K$; and
     ○ $\text{Listy}_i :=$ a list of K entries, $(i, j, \mathcal{F}(i, j))$, sorted into decreasing order of the $\mathcal{F}(i, j)$ values.
   - $Q :=$ the empty list.

1. Pop all entries off all Listy_i lists for which $\mathcal{F}(i, j) \geq (2^{\text{ctr}_i} - 1)\Delta$.

2. While Q not empty:
   a. Pop an entry, $(i, j, \mathcal{F}(i, j))$, off of Q.
   b. If (i, j) entry in $\mathcal{F}$ not yet crossed out:
      i. Cross out the (i, j) entry in $\mathcal{F}$.
      ii. Decrement the counter, ctr_i.
      iii. Pop all entries, $(i, j', \mathcal{F}(i, j'))$, from Listy_i for which $\mathcal{F}(i, j') \geq (2^{\text{ctr}_i} - 1)\Delta$; push them onto Q.
   c. Do Step b, above, for the entry, $(j, i, \mathcal{F}(j, i))$.

3. Let $UB = \max\{\text{ctr}_i\}$.

Table 3: Pseudo-code for the pre-processing algorithm

- ctr_i is an upper bound (initially $\text{ctr}_i = K$) on the number of distinct contingent time-points that can co-occur in any iSRN loop that contains C_i.

- Listy_i is a list of entries from the ith row of the $\mathcal{F}$ matrix, sorted into decreasing order.

As the algorithm runs, any entry, $(i, j, \mathcal{F}(i, j))$ from Listy_i, for which $\mathcal{F}(i, j) \geq (2^{\text{ctr}_i} - 1)\Delta$, signals that C_j could not occur in the same iSRN loop with C_i. Such entries are popped off Listy_i and pushed onto the global queue. As each entry from the global queue is processed, the corresponding ctr_i value decreases, which may lead to further entries moving from Listy_i to the global queue. The algorithm terminates whenever the global queue is emptied, at which point no further reductions in ctr_i values can be made. The algorithm returns the maximum ctr_i value, which specifies the maximum number of distinct contingent time-points that can co-occur in any iSRN loop in the given STNU. The Appendix proves that the algorithm's worst-case running time is $O(N^3)$.

In best-case scenarios, the pre-processing algorithm will result in all off-diagonal entries in $\mathcal{F}$ being crossed out, implying that there can be no nesting of LCR paths in any iSRN loop. In such cases, it is only necessary to do a single, $O(N^3)$-time round of the N^4 algorithm to ascertain whether the STNU is dynamically controllable. The benefit in such cases can be dramatic, for if the network contains even one semi-reducible path having K levels of nesting, then the unaided N^4 algorithm would needlessly perform K rounds of processing in $O(N^4)$ time.

6 CONCLUSIONS

This paper presented a new way of analyzing the structure of STNU graphs with the aim of speeding up DC checking. It proved that the number of occurrences of lower-case edges in any iSRN loop is bounded above by $2^K - 1$. It presented a recursive algorithm for constructing STNUs that contain iSRN loops that attain this upper bound, thereby showing that the bound is tight. In view of their highly convoluted structure, such loops are called *magic loops*. Finally, it presented an $O(N^3)$-time pre-processing algorithm that exploits the $2^K - 1$ bound to speed up DC checking for some networks. Thus, the paper makes both theoretical and practical contributions.

Other researchers have sought to speed up the process of DC checking using incremental algorithms. Stedl and Williams (2005) developed *Fast-IDC*, an incremental algorithm that maintains the dispatchability of an STNU after the insertion of new constraints or the tightening of existing constraints. Shah et al. (2007) extended *Fast-IDC* to accommodate the removal or weakening of constraints. Although intended to be applied incrementally, their algorithm showed orders of magnitude improvement over an earlier pseudo-polynomial DC-checking algorithm when evaluated empirically, checking dynamic controllability from scratch. It would be interesting to see if their work could be applied to generate an incremental version of the Morris' N^4 algorithm.

Others have extended the concept of dynamic controllability to accommodate various combinations of probability, preference and disjunction. For example, Tsamardinos (2002) augmented contingent durations with probability density functions and provided a method that, under certain restrictions, finds "the schedule that maximizes the probability of executing the plan in a way that respects the temporal constraints." Tsamardinos et al. (2003) then extended that work by developing algorithms to compute lower and upper bounds for the probability of a legal plan execution. Morris et al. (2005) similarly used probability density functions to represent the uncertainties associated with contingent durations, but also incorporated preferences over event durations. Rossi et al. (2006) presented a thorough treatment of STNUs augmented with preferences (but not probabilities). They defined the *Simple Temporal Problem with Preferences and Uncertainty* (STPPU) and notions of weak, strong and dynamic controllability.

Effinger et al. (2009) defined dynamic controllability for *temporally-flexible reactive programs* that include the following constructs: "conditional execution, iteration, exception handling, non-deterministic choice, parallel and sequential composition, and simple temporal constraints". They presented a DC-checking algorithm for temporally-flexible reactive programs that frames the problem as an "AND/OR search tree over candidate program executions."

REFERENCES

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49:61–95.

Effinger, R., Williams, B., Kelly, G., and Sheehy, M. (2009). Dynamic controllability of temporally-flexible reactive programs. In Gerevini, A., Howe, A., Cesta, A., and Refanidis, I., editors, *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 09)*. AAAI Press.

Hunsberger, L. (2010). A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME-2010)*, pages 121–128, Los Alamitos, CA, USA. IEEE Computer Society.

Morris, P. (2006). A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming (CP 2006)*, volume 4204 of *Lecture Notes in Computer Science*, pages 375–389. Springer.

Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In Nebel, B., editor, *17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 494–499. Morgan Kaufmann.

Morris, R., Morris, P., Khatib, L., and Yorke-Smith, N. (2005). Temporal constraint reasoning with preferences and probabilities. In Brafman, R. and Junker, U., editors, *Proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, pages 150–155.

Rossi, F., Venable, K. B., and Yorke-Smith, N. (2006). Uncertainty in soft temporal constraint problems: A general framework and controllability algorithms for the fuzzy case. *Journal of Artificial Intelligence Research*, 27:617–674.

Shah, J., Stedl, J., Robertson, P., and Williams, B. C. (2007). A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In Mark Boddy et al., editor, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. AAAI Press.

Stedl, J. and Williams, B. C. (2005). A fast incremental dynamic controllability algorithm. In *Proceedings of the ICAPS Workshop on Plan Execution: A Reality Check*, pages 69–75.

Tsamardinos, I. (2002). A probabilistic approach to robust execution of temporal plans with uncertainty. In Vlahavas, I. P. and Spyropoulos, C. D., editors, *Proceed-*

ings of the Second Hellenic Conference on AI (SETN 2002), volume 2308 of *Lecture Notes in Computer Science*, pages 97–108. Springer.

Tsamardinos, I., Pollack, M. E., and Ramakrishnan, S. (2003). Assessing the probability of legal execution of plans with temporal uncertainty. In *Proceedings of the ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information*.

Vidal, T. and Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In Wahlster, W., editor, *12th European Conference on Artificial Intelligence (ECAI-96)"*, pages 48–54. John Wiley and Sons, Chichester.

APPENDIX: Proof Sketches

Lemma 1 Proof. Each extension sub-path consists of zero or more loops of length zero followed by a nice suffix. By the definition of an ESP, each of those building blocks satisfies the *positive-proper-prefix* and *negative-proper-suffix* properties. Thus, any LC edge belonging to one of those building-block sub-paths must have an extension sub-path within that same building block. Thus, if $\mathcal{P}_1$ and $\mathcal{P}_2$ are ESPs within a single path $\mathcal{P}$ whose intersection is non-empty, then one must be contained within the other. □

Theorem 3 Proof. Suppose $\mathcal{P}$ is a path such that every LC edge in $\mathcal{P}$ has a usable moat edge. By Lemma 1, extension sub-paths in $\mathcal{P}$ are either disjoint or nested. Consider an LC edge e whose ESP $\mathcal{P}_e$ is innermost (i.e., does not contain any other ESP). Then $\mathcal{P}_e$ must be an OU-path. By definition of ESP, every proper prefix of $\mathcal{P}_e$ has non-negative length and, thus, any upper-case edges they might reduce to can have their labels removed by the Label-Removal rule. Thus, $\mathcal{P}_e$ itself can be reduced to a single OU-edge, e'. Since $\mathcal{P}_e$'s moat edge is usable, $|e'| > -x$ and, thus, if e' is a breach, its upper-case label can be removed by the Label-Removal rule. Continuing this process recursively will reduce away all LC edges from $\mathcal{P}$. Thus, $\mathcal{P}$ is semi-reducible.

Going the other way, suppose that $\mathcal{P}$ is a path with at least one LC edge, e, that does not have a usable moat edge. If e does not have an ESP, then e cannot be reduced away. Alternatively, if e has an ESP, $\mathcal{P}_e$, whose moat edge, e_m is unusable, then $|\mathcal{P}_e| < -x$ and e_m is a breach. In that case, $\mathcal{P}_e$ will reduce away to a UC edge whose label cannot be removed by the Label-Removal rule, thereby preventing e from being reduced away. □

Corollary 1 Proof. The usability of a moat edge cannot be affected by extracting a pesky prefix from an ESP. □

Corollary 2 Proof. The recursive procedure used in the first part of the proof of Theorem 3 shows how to use the corresponding extension sub-paths to reduce away any LC edges from a semi-reducible path, starting with innermost ESPs and working outward. □

Lemma 2 Proof. If $\mathcal{P}$ is an SRN loop that is not indivisibile, then it has an SRN sub-loop Q having fewer occurrences of LC edges. Recursively extracting SRN sub-loops in this way cannot continue for more than $\#\mathcal{P}$ steps and, thus, must eventually yield an iSRN loop. □

Theorem 4 Proof. Let $\mathcal{P}$ be an iSRN loop in an STNU having K contingent links. Let $e_1, \ldots, e_n$ be the root-level LC edges in $\mathcal{P}$; and $\ell_1, \ldots, \ell_n$ the corresponding root-level LCR sub-paths. By Lemma 1, these sub-paths are disjoint. By Corollary 2, these LCR sub-paths can be reduced to core edges, $E_1, \ldots, E_n$. Let $\mathcal{P}'$ be the OU-path obtained from $\mathcal{P}$ by replacing the root-level LCR sub-paths by their corresponding core edges. If any e_i and e_j were occurrences of the same LC edge, then $\mathcal{P}'$ could be split into sub-loops, $\mathcal{P}'_1$ and $\mathcal{P}'_2$, one containing E_i, the other containing E_j. Since $\mathcal{P}$ is an iSRN loop and $|\mathcal{P}| = |\mathcal{P}'_1| + |\mathcal{P}'_2|$, $|\mathcal{P}'_1| < 0$ or $|\mathcal{P}'_2| < 0$. Without loss of generality, suppose $|\mathcal{P}'_1| < 0$. Replacing the core edges in $\mathcal{P}'_1$ by the corresponding LCR sub-paths yields an SRN sub-loop of $\mathcal{P}$ having fewer occurrences of LC edges, contradicting that $\mathcal{P}$ is an iSRN loop. □

Interlude. Morris (2006) used a technique of extracting a sub-loop from a semi-reducible path while preserving its semi-reducibility. The following lemma specifies general conditions that suffice for such an operation to succeed.

Extra Lemma A. Let $\mathcal{P}$ be a semi-reducible path that contains a sub-loop, S, such that: (1) $|S| \geq 0$; (2) every ESP in $\mathcal{P}$ that contains S, but whose corresponding LC edge is not in S, is breach-free; and (3) S does not contain any suffix of any ESP in $\mathcal{P}$ whose corresponding LC edge is not in S. Then the path, $\mathcal{P}'$, formed by extracting S from $\mathcal{P}$ is semi-reducible with $|\mathcal{P}'| \leq |\mathcal{P}|$.

Proof of Extra Lemma A. Since $|S| \geq 0$, $|\mathcal{P}'| \leq |\mathcal{P}|$. Thus, it only remains to show that $\mathcal{P}'$ is semi-reducible. Let e be any LC edge in $\mathcal{P}'$. Since e also belongs to $\mathcal{P}$, and $\mathcal{P}$ is semi-reducible, it follows that e must have an ESP, $\mathcal{P}_e$, in $\mathcal{P}$ that can be used to reduce away e in $\mathcal{P}$.

Now, since e is in $\mathcal{P}'$, e cannot be in S. Thus, by condition (3), S cannot contain any suffix of $\mathcal{P}_e$. Thus, either $\mathcal{P}_e$ does not intersect S or $\mathcal{P}_e$ contains all of S. In the first case, $\mathcal{P}_e$ is a sub-path of $\mathcal{P}'$ and thus can be used to reduce away e in $\mathcal{P}'$ too. Hence, e has a usable moat edge in $\mathcal{P}'$. In the second case, the moat edge and possibly some of its predecessor edges in $\mathcal{P}_e$ cannot be contained in S. Now, extracting the non-negative sub-loop S from $\mathcal{P}_e$ may cause an earlier edge in $\mathcal{P}_e - S$ to become the new moat edge for e in $\mathcal{P}'$. However, by condition (2), $\mathcal{P}_e$ cannot contain any breaches; hence the moat edge for e in $\mathcal{P}_e - S \subseteq \mathcal{P}'$ must be usable. Since e was an arbitrary LC edge in $\mathcal{P}'$, $\mathcal{P}'$ must be semi-reducible by Theorem 3. □

Theorem 5 Proof. First, if $\mathcal{P}$ contains a breach, let $\mathcal{P}_e$ be an *outermost* breach-containing ESP for some LC edge e corresponding to a contingent link, (A, x, y, C). (Recall that ESPs must be nested or disjoint.) Let B_e be the rightmost breach edge in $\mathcal{P}_e$. Then B_e is a UC edge labeled by C and terminating in the activation time-point A. Let S be the sub-loop from the starting time-point of the LC edge e to the ending time-point of the UC edge B_e. Since S is a prefix of $\mathcal{P}_e$, $|S| \geq 0$. Let $\mathcal{P}'$ be the path obtained from $\mathcal{P}$ by extracting the sub-loop S. Conditions (2) and (3) of the Extra Lemma can also be shown to hold for S. Thus, $\mathcal{P}'$ must be an SRN loop containing fewer occurrences of LC edges that $\mathcal{P}$, contradicting that $\mathcal{P}$ is an iSRN loop.

Next, suppose the depth of nesting of ESPs in $\mathcal{P}$ is more than K. Then $\mathcal{P}$ must have an occurrence of some LC edge e whose corresponding ESP, $\mathcal{P}$, contains another occurrence of e. Let S be the prefix of $\mathcal{P}_e$ whose final edge is that second

occurrence of e. Then S is a non-negative sub-loop. Let $\mathcal{P}'$ be as in the Extra Lemma. Conditions (2) and (3) of the Extra Lemma can then be shown to hold. Thus, $\mathcal{P}'$ must be an iSRN loop having fewer occurrences of LC edges than $\mathcal{P}$, contradicting that $\mathcal{P}$ is an iSRN loop. $\square$

Theorem 6 Proof. Suppose, contrary to the theorem, that every LC edge in some iSRN loop $\mathcal{P}$ occurs more than once. Let e_1 be some root-level LC edge. By the proof of Theorem 5, e_1 cannot occur within its own root-level LCR sub-path; and, by the proof of Theorem 4, e_1 cannot occur more than once at the root-level. Thus, a second occurrence of e_1 must occur in the interior of some other root-level LCR sub-path for some root-level LC edge e_2. Similarly, a second occurrence of e_2 must occur in the interior of some other root-level LCR sub-path for some root-level LC edge e_3. And so on. Since there are at most K distinct LC edges in $\mathcal{P}$, this pattern of inclusion must circle back on itself.

Let $\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_v$ be the root-level LC edges in this cycle, listed in their order of appearance in $\mathcal{P}$. (Since $\mathcal{P}$ is a loop, this listing is not unique.) Thus, $\hat{e}_2$ is contained in the root-level LCR sub-path for $\hat{e}_1$, $\hat{e}_3$ is contained in the root-level LCR sub-path for $\hat{e}_2$, and so on. Chop $\mathcal{P}$ into $2v$ pieces, $\rho_1, \sigma_1, \rho_2, \sigma_2, \ldots, \rho_v, \sigma_v$, where the first edge of each ρ_i is the LC edge $\hat{e}_i$; and the first edge of each σ_i is the occurrence of $\hat{e}_{i-1}$ in the LCR sub-path for $\hat{e}_i$. (We take $\hat{e}_0$ to represent $\hat{e}_v$.) By construction, $|\rho_i| > 0$; thus, $\sum |\rho_i| > 0$.

Next, for each i, let ψ_i be the sub-path whose first edge is the interior occurrence of $\hat{e}_i$ and whose last edge is the edge in $\mathcal{P}$ that precedes the root-level occurrence of e_i. By construction, each ψ_i is a semi-reducible loop with $\#\psi_i < \#\mathcal{P}$; thus, since $\mathcal{P}$ is an iSRN loop, $|\psi_i| \geq 0$ for each i and $\sum |\psi_i| \geq 0$. Finally, by permuting these sub-paths, it can be shown that $\sum |\psi_j| = \sum |\sigma_j| + m|\mathcal{P}|$ for some integer $m > 0$. But then $|\mathcal{P}| \geq 0$, contradicting that $\mathcal{P}$ is an iSRN loop. $\square$

Theorem 7 Proof. By induction on K. For the base case, $K = 0$ and thus there are $2^0 - 1 = 0$ occurrences of LC edges in every iSRN loop. For the recursive case, suppose that every iSRN loop in an STNU with K contingent links has at most $2^K - 1$ occurrences of LC edges. Suppose that $\mathcal{P}$ is an iSRN loop in an STNU with $K + 1$ contingent links. By Theorem 6, $\mathcal{P}$ must contain some LC edge, e, that occurs at most once in $\mathcal{P}$, and that that occurrence must be at the root level. Let ℓ be the root-level LCR sub-path for e in $\mathcal{P}$; and let E be the core edge that ℓ can be transformed into.

Let $\mathcal{P}_1$ be the path obtained from $\mathcal{P}$ by replacing ℓ by the core edge E. Since the only occurrence of e in $\mathcal{P}$ was in ℓ, e does not occur in $\mathcal{P}_1$. Next, let $\mathcal{P}_1'$ be obtained from $\mathcal{P}_1$ by removing any upper-case labels from its upper-case edges that match the lower-case label on e. By construction, $\mathcal{P}_1'$ does not contain e or any matching UC edges; thus, it can be viewed as a path in an STNU having only K contingent links. Furthermore, $|\mathcal{P}_1'| = |\mathcal{P}| < 0$. It can be shown that $\mathcal{P}_1'$ is an iSRN loop in an STNU having K contingent links; and therefore that $\#\mathcal{P}_1' \leq 2^K - 1$.

For the next part of the proof, let e be the root-level LC edge described above; and let $\mathcal{P}_e$ be its root-level ESP. Then, let $\mathcal{P}_2$ be the loop formed by $\mathcal{P}_e$ and a single *new* edge e_α. By construction, if $\mathcal{P}_e$ is a sub-path from C to some time-point X, then e_α is a new edge from X back to C. The length of this new edge, e_α, is determined as follows:

- If $\mathcal{P}_e$ does not have any negative-length sub-loops, then

$|e_\alpha| = |\mathcal{P}| - |\mathcal{P}_e|$ (i.e., the length of the portion of $\mathcal{P}$ that e_α is effectively replacing).

- Otherwise, let $|e_\alpha| = M - |\mathcal{P}_e|$, where $M = \max\{|S| : S$ is a negative sub-loop of $\mathcal{P}_e\}$.

By construction, the only occurrence of e in $\mathcal{P}$ has been extracted and, thus, does not appear in $\mathcal{P}_2$. In addition, since $\mathcal{P}$ is an iSRN loop, $\mathcal{P}_e$ cannot have any breaches; thus, there are no occurrences of UC edges in $\mathcal{P}_e$ labeled by C. Thus, $\mathcal{P}_2$ can be viewed as a loop in an STNU having K contingent links. It can be shown that $\mathcal{P}_2$ is an iSRN loop; and thus that $\#\mathcal{P}_2 \leq 2^K - 1$. Finally,

$$\#\mathcal{P} = \leq (2^K - 1) + 1 + (2^K - 1) = 2^{(K+1)} - 1. \quad \square$$

Theorem 8 Proof. Let $\mathcal{P}$ be an iSRN loop. By Theorem 5, $\mathcal{P}$ must be breach-free. Suppose that $\mathcal{P}$ contains an occurrence of an ordinary edge, $A \xrightarrow{\ y\ } C$, associated with a contingent link, (A, x, y, C). Call this edge e°. Let e be the corresponding LC edge, $A \xrightarrow{\ c:x\ } C$. We shall show that $\mathcal{P}$ can be converted into an SRN loop $\mathcal{P}^\dagger$ by replacing or removing the offending occurrence of e°.

First, let $\mathcal{P}_x$ be the same as $\mathcal{P}$ except that the occurrence of e° has been replaced by the LC edge e. If $\mathcal{P}_x$ is semi-reducible, then let $\mathcal{P}^\dagger = \mathcal{P}_x$. Otherwise, the non-semi-reducibility of $\mathcal{P}_x$ must be due to the inclusion of the LC edge e. That LC edge must have an unusable moat edge (i.e., it must have a breach moat edge, B_e, with $|\mathcal{P}_e| < -x$). This implies that there must be a sub-loop, S, from A to A in $\mathcal{P}$. The first edge of S is e°; its last edge is B_e. Let $\mathcal{P}^\dagger$ be obtained by extracting the sub-loop S from $\mathcal{P}$. Since S contains the offending occurrence of e°, it follows that $\mathcal{P}^\dagger$ does not. The conditions of Extra Lemma A can be shown to hold for S. Thus, $\mathcal{P}^\dagger$ is an SRN loop. If $\mathcal{P}^\dagger$ happens not to be indivisible, then SRN sub-loops can be extracted (as in the proof of Lemma 2) until indivisibility is restored. Recursively removing any remaining ordinary edges associated with contingent links produces the result.

The case of the other kind of ordinary edge (cf. Fig. 1) associated with contingent links is handled similarly. $\square$

Note. Because Theorems 9 and 10 are not needed for the rest of the paper, their proofs are omitted for space reasons.

Proof that pre-processing algorithm runs in $O(N^3)$ time. Step 0, which happens only once, is dominated by the computation of the OU-distance matrix, $\mathcal{D}^\diamond$, which can be done in $O(N^3)$ time, for example, using the Floyd-Warshall algorithm (Cormen et al., 2009). Sorting K lists of K entries can be done in $O(K^2 \log K)$ time, for example, using K runs of the MergeSort algorithm (Cormen et al., 2009).

Step 1, which happens only once, involves scanning no more than K^2 entries; thus, it can be done in $O(K^2)$ time.

The *while* loop in Step 2 runs at most $K(K-1)/2$ times. All subsidiary steps, except Step 2b(iv) can be done in constant time. Thus, ignoring Step 2b(iv), all of the iterations of the *while* loop in Step 2 can be done in $O(N^2)$ time. As for Step 2b(iv), each entry in the Listy_i lists can only be popped/pushed once; thus, the total number of pop/pushes from Step 2b(iv) is order $O(K^2)$. In addition, each time through Step 2b(iv), the stopping condition is hit only once. Thus, the total number of stoppings in Step 2b(iv) is $O(K^2)$. Thus, Step 2, in its totality, is $O(K^2)$.

Since Step 3 is $O(K)$, the overall worst-case time complexity of the pre-processing algorithm is $O(N^3)$.