IMPROVED ALGORITHMS FOR ALLEN'S INTERVAL ALGEBRA BY DYNAMIC PROGRAMMING WITH SUBLINEAR PARTITIONING

LEIF ERIKSSON AND VICTOR LAGERKVIST

ABSTRACT. *Allen's interval algebra* is one of the most well-known calculi in qualitative temporal reasoning with numerous applications in artificial intelligence. Recently, there has been a surge of improvements in the *fine-grained* complexity of NP-hard reasoning tasks, improving the running time from the naive $2^{\mathcal{O}(n^2)}$ to $\mathcal{O}^*((1.0615n)^n)$, with even faster algorithms for unit intervals a bounded number of overlapping intervals (the $\mathcal{O}^*(\cdot)$ notation suppresses polynomial factors). Despite these improvements the best known lower bound is still only $2^{o(n)}$ (under the *exponential-time hypothesis*) and major improvements in either direction seemingly require fundamental advances in computational complexity. In this paper we propose a novel framework for solving NP-hard qualitative reasoning problems which we refer to as *dynamic programming with sublinear partitioning*. Using this technique we obtain a major improvement of $\mathcal{O}^*((\frac{cn}{\log n})^n)$ for Allen's interval algebra. To demonstrate that the technique is applicable to more domains we apply it to a problem in qualitative spatial reasoning, the *cardinal direction point algebra*, and solve it in $\mathcal{O}^*((\frac{cn}{\log n})^{2n/3})$ time. Hence, not only do we significantly advance the state-of-the-art for NP-hard qualitative reasoning problems, but obtain a novel algorithmic technique that is likely applicable to many problems where $2^{\mathcal{O}(n)}$ time algorithms are unlikely.

1. INTRODUCTION

Allen's interval algebra (ALLEN) is one of the most well-known examples of qualitative temporal reasoning where the task is to decide whether a set of (typically incompletely specified) intervals in $\mathbb{R}^2$ can be ordered in a consistent way. ALLEN is an influential formalism with a wide range of applications in artificial intelligence, and for a handful of concrete examples we may e.g. mention planning [1, 4, 12, 13], natural language processing [3, 14], and molecular biology [9]. The second major problem that we consider, the *cardinal direction point algebra* (CDPA), can be seen as an offshoot of Allen's interval algebra where the task is to reason about directions of objects in a 2-dimensional space. This algebra has seen applications in e.g. geographical information science and image retrieval and we refer the reader to the survey by Dylla et al. [5] for additional references and applications for both these formalisms.

Both these formalisms are in general NP-hard and admit non-trivial tractable fragments. Hence, in actual applications one is rarely contempt with reasoning only with tractable fragments, which spurs the question of how fast we can solve NP-hard qualitative reasoning tasks. First, we need to carefully establish a baseline upper bound that we can use to measure whether an algorithm for these problems is indeed an improvement or not. This is easy for many types of problems over *finite* universes, e.g., finite domain *constraint satisfaction problems* (CSPs) and the Boolean satisfiability problem, which can always be solved simply by enumerating all possible assignments from variables to the finite universe. Clearly, this cannot be done for ALLEN or CDPA, but these problems can be solved in $2^{\mathcal{O}(n\cdot\log n)}$ time by enumerating certain total orderings, which for ALLEN gives the precise upper bound $\mathcal{O}^*((2n)^{2n})$ [10][1]. Hence, the question is whether this upper bound can be improved, and we stress that this is sometimes known to be possible (e.g., k-SAT and k-COLORING) but that there also exist problems where this is conjectured to be impossible (e.g., CNF-SAT, SAT with unrestricted clause length). However, this has been proven possible for ALLEN which was recently solved in $\mathcal{O}^*((1.0615n)^n)$ time [6] by a novel dynamic programming algorithm. Additionally, if ALLEN is restricted to

[1]The notation $\mathcal{O}^*(\cdot)$ suppresses polynomial factors and n denotes the number of intervals.

intervals of length one, then it can even be solved in $2^{\mathcal{O}(n \log \log n)}$ time [2]. A faster $\mathcal{O}^*(c^n)$ time algorithm is also known for the special case where no point occurs inside more than k intervals [7]. However, despite these improvements, we are still far away from an unconditional single-exponential $\mathcal{O}^*(c^n)$ time algorithm and even further away from the best-known lower bounds which only rule out subexponential algorithms running in $2^{o(n)}$ time under the *exponential-time hypothesis* [11].

In this paper we make a significant push for ALLEN and CDPA by introducing a new variant of dynamic programming; *dynamic programming with sublinear partitioning*, which is exceptionally suited for problems related to orderings. After having introduced the necessary preliminaries (in Section 2) we first showcase our new method for ALLEN in Section 3 where we obtain the bound $\mathcal{O}^*((\frac{cn}{\log n})^n)$, for some constant $c \geq 1$. This is a major improvement compared to the aforementioned bound of $\mathcal{O}^*((1.0615n)^n)$ and thus proves that ALLEN is solvable in $\mathcal{O}^*(f(n)^n)$ time for $f \in o(n)$, i.e., a significant leap towards a $\mathcal{O}^*(c^n)$ time algorithm. To exemplify that our method is not limited to ALLEN we continue (in Section 4) by showing how it can be tailored to solve CDPA faster than the general case of ALLEN. Here, we obtain the bound $\mathcal{O}^*((\frac{cn}{\log n})^{2n/3})$ for a constant c, which is *much* faster than the algorithm for ALLEN. The main idea behind our dynamic programming strategy is to use a table consisting of records, representing (partially) placed start- and end-points of intervals, where the twist is that the size of a record is not fixed beforehand but may expand up to a certain size during the execution of the algorithm. To achieve this we use a similar idea to the one presented in Eriksson & Lagerkvist [6] but we analyze the structure of potential solutions much deeper. We manage to identify redundant information by using the fact that if we know the relative position of more variables then we need to keep track of fewer potential positions. While previous approaches have required keeping track of, or storing information corresponding to, a linear number of partitions measured in the number of variables, we here manage to solve the task while only partitioning the variables into a sublinear number of partitions. Consequently, the upper limit of information stored in a record is bounded tighter by the number of variables than any earlier approach. In turn, this leads to a lower number of maximum records, and hence a lower overall complexity. This idea of managing with only a sublinear number of partitioning will be our main result throughout the paper. However, achieving this is as expected far from trivial.

Last, it is worth observing that virtually all qualitative temporal and spatial reasoning problems can be formulated as CSPs over a template containing a strict partial order or an acyclic relation [11]. Hence, all members of this diverse and rich class of problems are intrinsically related to the existence of certain orders, and it is not a large stretch of the imagination to conjecture that more problems can be solved with our approach. Most pressingly: Can dynamic programming with sublinear partitioning be used to solve an NP-hard qualitative reasoning problem in $\mathcal{O}^*(c^n)$ time? We discuss this and additional questions in Section 5.

2. PRELIMINARIES

Given a set of finitary relations Γ defined on a (potentially infinite) set D of values, we define the constraint satisfaction problem over Γ (CSP(Γ)) as follows.

CSP(Γ)
Instance: A tuple (V, C), where V is a set of variables and C a set of constraints of the form $R(v_1, \ldots, v_t)$, where t is the arity of $R \in \Gamma$ and $v_1, \ldots, v_t \in V$.
Question: Is there a function $f \colon V \to D$ such that $(f(v_1), \ldots, f(v_t)) \in R$ for every $R(v_1, \ldots, v_t) \in C$?

The set Γ is referred to as a *constraint language*, while the function f is a function *satisfying* the instance I, or simply a *model* of I. Given an instance I of CSP(Γ), we let $||I||$ denote the number of bits required to represent I.

We continue by defining total orders and order relations induced by a given total order. A total order $(S, \leq)$ is a relation $\leq$ on set S which is *reflexive* (for all $x \in S$, $x \leq x$), *antisymmetric* (for all $x, y \in S$, if $x \leq y$ and $y \leq x$, then $x = y$), *transitive* (if $x \leq y$ and $y \leq z$, then $x \leq z$), and *strongly connected* (for all $x, y \in S$, either $x \leq y$ or $y \leq x$). If $\odot \in \{<, >, =\}$ and $T = (S, \leq_T)$ is a total order then we write $\odot_T$ for the relation

$\boldsymbol{Relation}$	$\boldsymbol{Illustration}$	$\boldsymbol{Interpretation}$
$\boldsymbol{X < Y}$ $\boldsymbol{Y > X}$	XXX YYY	X precedes Y
$\boldsymbol{X = Y}$	XXXXXXX YYYYYYY	X is equal to Y
$\boldsymbol{X\ m\ Y}$ $\boldsymbol{Y\ mi\ X}$	XXX YYY	X meets Y
$\boldsymbol{X\ o\ Y}$ $\boldsymbol{Y\ oi\ X}$	XXXXX YYYY	X overlaps with Y
$\boldsymbol{X\ s\ Y}$ $\boldsymbol{Y\ si\ X}$	XXX YYYYYYY	X starts Y
$\boldsymbol{X\ d\ Y}$ $\boldsymbol{Y\ di\ X}$	XXX YYYYYYY	X during Y
$\boldsymbol{X\ f\ Y}$ $\boldsymbol{Y\ fi\ X}$	XXX YYYYYYY	X finishes Y

TABLE 1. The 13 basic relations between two intervals on the same line. (i denotes the $i$nverse/converse of a relation.)

induced by T: $x <_T y$ if $x \leq_T y$ and $y \leq_T x$ does not hold, conversely for $>_T$, and $x =_T y$ if $x \leq_T y$ and $y \leq_T x$. The total number of different total orders for a variable set of n elements is known as n'th *Ordered Bell Number* ($OBN(n)$) and is in $\mathcal{O}^*((\frac{n}{e \log 2})^n)$ and $\mathcal{O}^*((0.5307n)^n)$ (recall that the $O^*(\cdot)$ notation supresses polynomial factors).

2.1. **Allen's Interval Algebra.** In *Allen's interval algebra* the basic relations consists of 13 relations between intervals over $\mathbb{R}^2$ (summarized in Table 1). The computational problem is then defined as follows.

ALLEN'S INTERVAL ALGEBRA

Instance: A set of intervals V and a set of binary constraints C of the form $c(X, Y)$ where $c \subseteq \{<, >, =, m, mi, o, oi, s, si, d, di, f, fi\}$ and $X, Y \in V$.
Question: Is there a total order $T = (S, \leq)$ and a function $f\colon V \to S^2$ such that for all $X \in V$ with $f(X) = (u, v)$ then $u < v$ and for every constraint $c(X, Y) \in C$ then $f(X) \odot f(Y)$ for some $\odot \in c$?

Equivalently, the problem can be defined as CSP($\mathcal{A}$) where $\mathcal{A}$ contains the 13 basic relations in Table 1 and is closed under union. Note from Table 1 that any relation between two intervals can be described using point relations ($<$, $>$ and $=$) between the start- and end-point of the two intervals. For a set of intervals V we write V^- for the set of start-points and V^+ for the set of end-points.

2.2. **Cardinal Direction Point Algebra.** We follow the projection based approach of Frank [8] and define the second major problem of the paper as follows.

CARDINAL DIRECTION POINT ALGEBRA

Instance: A set of pairs of points, V, and a set of binary constraints C, of the form $c(u, v)$, where $c \subseteq \{\{<, >, =\} \times \{\{<, >, =\}\}$ and $u, v \in V$.
Question: Is there a function $f\colon V \to \{1, \ldots, |V|\}^2$ such that for every constraint $c(u, v) \in C$ then $f(u) = (u_x, u_y)$, $f(v) = (v_x, v_y)$, $u_x \odot_x v_x$ and $u_y \odot_y v_y$ with $(\odot_x, \odot_y) \in c$?

Alternatively, the problem can be seen as whether V can be ordered into two separate total orderings such that one represents the x-dimension and the other the y, such that they combined satisfy all constraints.

Another alternative is viewing CDPA as a sub-problem of ALLEN (e.g. by letting a^- represent a_x and a^+ represent the a_y coordinate of a point a).

3. A NEW ALGORITHM FOR ALLEN'S INTERVAL ALGEBRA

In this section we present a novel, improved algorithm for ALLEN based on dynamic programming. The twist with this algorithm is that the dimension of our table, or the size of the records stored, is not static, but may dynamically change during the execution of the algorithm. This dynamic programming is realized by using records functioning as elements or keys in a map that is dynamically built, and we refer to this programming scheme as *dynamic programming with sublinear partitioning*. Here, sublinear refers to the fact that we partition the variable set into a sublinear number of partitions. Each such record represents a set of S variables that have been used to construct this record, and then pairs of disjoint subsets of S, and Booleans. The subsets of S are here used to keep track of the (relevant) ordering between variables that have been used, while the Booleans serve as markers that tell us if a certain subset has ever been merged with another subset or not. This is useful, since if they have previously been merged with some other subset, we do not want to add any new variables to this subset, since we could then get inconsistent cases where $x = y$, $y = z$ but $x < z$. Combined, these partitions of S represent an ordered partition, but we will not strictly use them as such.

We also introduce two notations for records: *left-heaviness* where we require start-points of intervals in a record to always be in either of the first two subsets, and *pair-placing* where we require whole intervals, and not just a start- or an end-point, to be placed at the same time into a record. Formally, these notions are now defined as follows.

Definition 1. *For a CSP($\mathcal{A}$) instance (V, C) we say that $r = (S, (S_1, B_1), \ldots, (S_m, B_m))$ is a* Record *if the following holds:*

1. $S \subseteq V^- \cup V^+$,
2. $B_1, \ldots, B_m \in \{0, 1\}$, *and*
3. $S_1, \ldots, S_m$ *is a partitioning of* S.

We say that m is the size of the record. A record is left-heavy *if $S_i \cap V^- = \emptyset$ for all $3 \leq i \leq m$. A record is* pair-placing *if and only if for all $x^- \in S_i$ for any i, there is a $j > i$ such that $x^+ \in S_j$ or $i = j$ and $B_i = 0$.*

For any two variables u, v in some Record with $u \in S_i$ and $v \in S_j$, then we also assume that $u < v$ if $i < j$, $u > v$ if $i > j$ and that $u = v$ if $i = j$ and $B_i = 1$. However, if $i = j$ and $B_i = 0$ we have to assume all three relations between u and v holds. Further, a Record contradicts *C if and only if there is a constraint $c(X, Y) \in C$ such that no assumption we can make about the relations X and Y given said Record will satisfy $c(X, Y)$.*

Generalization between records, i.e. whether a record A can be converted to a record B by adding variables and merging subsets, will be important, and we define the following concept.

Definition 2. *We say that for two records*

$r_1 = (S_1, (S_{1,1}, B_{1,1}), \ldots, (S_{1,m}, B_{1,m}))$

and

$r_2 = (S_2, (S_{2,1}, B_{2,1}), \ldots, (S_{2,k}, B_{2,k}))$

then r_2 is a generalization of r_1 if:

1. $S_1 \subseteq S_2$,
2. $k \leq m$ *(r_2 is not larger than r_1),*
3. *for every $S_{1,i}$ there is a $S_{2,j}$ such that $S_{1,i} \subseteq S_{2,j}$ (no set in r_1 is split in r_2),*
4. *for every j then $B_{2,j} = 1$ if and only if either $S_{2,j} = \emptyset$ or if there is an i such that $B_{1,i} = 1$ and $S_{1,i} = S_{2,j}$, otherwise $B_{2,j} = 0$ (only empty or unchanged sets keep their positive B value), and*

(5) *for all variables $x \in S_{1,i}$ and $x' \in S_{1,i'}$ with $i \leq i'$ then $x \in S_{2,j}$ and $x' \in S_{2,j'}$ such that $j \leq j'$ (ordering is kept).*

For a record $r = (S, (S_1, B_1), \ldots, (S_m, B_m))$ we can recursively generate all generalizations $r' = (S, (S'_1, B'_1), \ldots, (S'_{m'}, B'_{m'}))$ with $m' \leq m$ in $\mathcal{O}^*(3^m)$ time. We do this by branching on every S_i and constructing two new records, one where S_i is merged with S_{i-1} (when these exists) and (S_{i-1}, B_{i-1}) is removed and one where we also replace (S_{i-1}, B_{i-1}) with $(\emptyset, 1)$. For the value of B'_i we strictly follow the definition. By checking if a given result has earlier already been created we can also prevent calling the recursion with the same record more than once. Our complexity of $\mathcal{O}^*(3^m)$ then comes from that either $(S'_i, B'_i) = (S_i, B_i)$, $(S'_{i-1}, B'_{i-1}) = (S_{i-1}, B_{i-1}) \cup (S_i, B_i)$ or $(S'_i, B'_i) = (S_{i-1}, B_{i-1}) \cup (S_i, B_i)$ and $(S'_{i-1}, B'_{i-1}) = (\emptyset, 1)$, creating fewer than 3^m different possibilities. Let $\mathrm{GenerateGeneralizations}(r, H)$ be the function doing this for a record r and where H is the set of all previously generated records. Note that the records in H does not need to be generalizations of r, since this set is just used to prevent unessesary computational complexity.

After this we want to continue by adding new intervals to existing records. When adding new intervals to records we do so in such a manner that given an ALLEN instance $I = (V, C)$ we never contradict C, while, if given a left-heavy and pair-placing record, the result is also left-heavy and pair-placing.

We briefly remind the reader that for an interval X the points x^- and x^+ represents the start- and endpoint of said interval.

Algorithm 1 Add a new pair to a given record R, assuming $B_2 = 1$, not contradicting C

```
1: function ADD(I = (V, C),
              R = (S, (S_1, B_1), ..., (S_m, B_m)), Output)
2:     for every x^-, x^+ ∈ (V^- ∪ V^+) \ ⋃_{i=1}^m S_i
           and every 2 ≤ j ≤ m with B_j = 1 do
3:         if for all X' ∈ V with x'^- ∈ S_u and x'^+ ∈ S_v all
             c(X, X'), c(X', X) ∈ C are satisfied by
             x^- = 2, x^+ = j, x'^- = u and x'^+ = v then
4:             Output ←GenerateGeneralizations((
                 S ∪ {x^-, x^+}, (S_1, B_1), (S_2 ∪ {x^-}, 1), ...,
                 (S_j ∪ {x^+}, 1), ..., (S_m, B_m)), Output)
5:         end if
6:     end for
7:     return Output
8: end function
```

As we can see, Algorithm 1 guarantees that for any pair X that we add to a record $(S, (S_1, B_1), \ldots, (S_m, B_m))$, all constraints $c(X, X') \in C$ with $X' \in S$ are satisfied. Quite obviously the work done in Algorithm 1 (if we exclude the work done by the functio $\mathrm{GenerateGeneralizations}$) is also polynomial in terms of $||I|| + m$. By only ever adding x^- to S_2 the algorithm keeps left-heaviness if the input is left-heavy, and as we add one pair (x^-, x^+) at a time, the result is also pair-placing if the input is pair-placing. Here we also see how the Boolean variables B_i are used, as we only add new start- and end-points to sets with a Boolean set to 1. Combined with the definition of generalization this helps ensure we avoid situations where now $x = y$ and $x = z$, but $x < z$ in an earlier record which the current record is a generalization of.

With Algorithm 1 we now have everything we need for solving ALLEN. However, we want to optimize our method and hence want to limit our records to a certain size, i.e. limit our m. The choice of m needs to be big enough to ensure correctness of the algorithm, i.e. big enough so that we can always ensure that every pair can be placed into some record, while the number of already placed pairs increases. Choosing $m = 4n$ would trivially give us enough room to place every single variable, be it a start- or end-point, in any possible

ordering. To do better we first make the following observation: given an arbitrary ordering of $V^- \cup V^+$ and placing pairs following the ordering of the start-points, if we let k be the number of pairs yet to be placed, we need at most $2k$ pairs (S_i, B_i) for which $B_i = 1$. This leads to a situation where we have records where for every second pair, the Boolean is set to one while for the other pairs it is set to zero, while still allowing the placement of every remaining variable. If more than $2k + 1$ pairs with $B_i = 0$ exists, there must exist two pairs $(S_j, 0)$ and $(S_{j+1}, 0)$ and hence a generalization with $(S_j \cup S_{j+1}, 0)$, but which is otherwise identical. Hence we take $m = 4k + 1$, which also leads us to our main algorithm as presented in Algorithm 2.

Algorithm 2 Solving ALLEN by dynamic programming.

```
1:  function MAIN(I = (V, C))
2:      R ← {(∅, (S_1 = ∅, 1), ..., (S_{4n+3} = ∅, 1))}
3:      for every record
            r = (S, (S_1, B_1), ..., (S_m, B_m)) ∈ R do
4:          if |S| = 2n then
5:              return True
6:          else if m ≤ 4(n − |⋃_{i=1}^{m} S_i|) + 3 then
7:              R ←Add(I, r, R)
8:          end if
9:      end for
10:     return False
11: end function
```

In Algorithm 2 we see how we make use of this decreasing number of pairs (S, B) by ignoring any records containing more pairs that we think we will need. This algorithm works by iteratively calling Algorithm 1 with new records, until every record has been either tested, or we find a record containing all intervals (in which case we must have a satisfiable instance).

Lemma 3. *For an arbitrary CSP($\mathcal{A}$) instance I, Algorithm 2 returns True if and only if I is a satisfiable instance.*

Proof. We begin by proving completeness. Given a satisfiable instance $I = (V, C)$, we know there exists a function $f : V^- \cup V^+ \to \mathbb{N}$ such that f satisfies every constraint in C. Let $V^- = \{x_1^-, \ldots, x_n^-\}$ and $V^+ = \{x_1^+, \ldots, x_n^+\}$ such that $f(x_1^-) \leq f(x_2^-) \leq \cdots \leq f(x_n^-)$, i.e., each pair x_i^-, x_i^+ is assigned an index i according to some ordering of the start-points of the intervals. Define the sets $T_1, \ldots, T_n$ such that $x_j^\pm \in T_i$ if and only if there is no variable $v \in \{x_i^-, x_i^+, \ldots, x_n^-, x_n^+\}$ such that $f(v) < f(x_j^\pm)$. Furthermore, define the auxiliary function $\#_< : V^- \cup V^+ \times 2^{V^- \cup V^+} \to \mathbb{N}$ such that $\#_<(x_j^\pm, U)$ returns the number of variables y such that $f(y) < f(x_j^\pm)$ and $y \in U$. From this we define the functions $f_{\#,i} : V^- \cup V^+ \to \{1, \ldots, 4n + 1\}$:

$$f_{\#,i}(x_j^\pm) = 2 \cdot \#_<(x_j^\pm, \{x_i^-, x_i^+, \ldots, x_n^-, x_n^+\}) + \begin{cases} 2, & \text{if } x_j^\pm \in T_i, \\ 1, & \text{if } x_j^\pm \notin T_i. \end{cases}$$

Since $f_{\#,i}$ follows the same ordering as f, it is also a solution for $I[\{x_i^-, x_i^+, \ldots, x_n^-, x_n^+\}]$, i.e. the sub-problem of I where only variables $\{x_i^-, x_i^+, \ldots, x_n^-, x_n^+\}$ are considered. Additionally, for all intervals $Y \in V$ and $X_i \in \{T_i \cup \ldots \cup T_n\}$, $f_{\#,i}$ also satisfies any constraints $c(X_i, Y) \in C$. For each $i \in \{0, \ldots, n\}$ define the record

$$r_i = (\{x_1^-, x_1^+, \ldots, x_i^-, x_i^+\}, (S_{1,i}, B_{1,i}), \ldots, (S_{4(n-i)+3,i}, B_{4(n-i)+3,i}))$$

such that $x_j^{\pm} \in S_{i',i}$ if and only if $f_{\#,i}(x_j^{\pm},) = i'$ and $j \leq i$, and $B_{i',i} = 1$ if and only if $S_{i',i} \cap S_i = \emptyset$ and $B_{i',i} = 0$ otherwise. According to our definitions r_{i+1} is a generalization of r_i, and every record is both left-heavy and pair-placing. As such, Algorithm 2 generates all records r_i, $i \in \{0, \ldots, n\}$ starting from the original record constructed at line 2, which is equivalent to r_0. By doing so the algorithm must reach a record containing all variables (r_n), and hence returns $True$.

For soundness, assume that the algorithm returns $True$. Then there must be an ordering of the variables in V, $X_1, \ldots, X_n$, and sequence of records $r_0, \ldots, r_n$ such that

$$r_i = (\{x_1^-, x_1^+, \ldots, x_i^-, x_i^+\}, (S_{1,i}, B_{1,i}), \ldots, (S_{4(n-i)+3,i}, B_{4(n-i)+3,i}))$$

for all $i \in \{0, \ldots, n\}$ and such that

$$(\{x_1^-, x_1^+, \ldots, x_i^-, x_i^+\}, (S_{1,j} \setminus S', B_{1,j}), \ldots, (S_{4(n-j)+3,j} \setminus S', B_{4(n-j)+3,j}))$$

is a generalization of r_i for all $j \in \{i+1, \ldots, n\}$ when

$$S' = \{x_{i+1}^-, x_{i+1}^+, \ldots, x_j^-, x_j^+\}.$$

This holds because of how $GenerateGeneralizations$ together with Add constructs records. Since $B_{i,j}$ must be 1 for Add to add new a new pair to $S_{i,j}$, since (1) $B_{i,j}$ is set to 0 as soon as two sets are merged into $S_{i,j}$, and (2) every constraint $c(X_i, X_{i'}) \in C$ must be satisfied for all $i' < i$, for X_i to be added. Given this set, define the function $h : V^- \cup V^+ \to \{1, \ldots, 4n+3\}$ such that (assuming $i > i'$) if $x_i^{\pm} \in S_{j,i}$ and $x_{i'}^{\pm} \in S_{j',i}$ with $j \odot j'$ then $h(x_i^{\pm}) \odot h(x_{i'}^{\pm})$. Since every $c(X_i, X_j) \in C$ is satisfied for all $j < i$ when X_i is added (since h orders X_i according to its position when added), and since this is true for all i, h must satisfy C. Hence, h is a solution for I, implying that our algorithm is sound. With both soundness and completeness proven, the algorithm must also be correct, and hence we are done. □

With all supporting lemmas in place, we can now present the main result of the section.

Theorem 4. *There is a constant c such that* ALLEN *can be solved in $\mathcal{O}^*((\frac{cn}{\ln n})^n)$ time and space using dynamic programming with sublinear partitioning.*

Proof. We know that Algorithm 2 is correct from Lemma 3. For the run-time we note that Add is polynomial (with respect to n and m), $GenerateGeneralizations$ adds fewer than 3^m sets to $Output$ and does linear work in terms of the size of the return. Our $Main$ itself also only does polynomial work for each record, and hence the question that remains is the maximal possible number of records. With the way we limit records size, we have that for a certain m there are at most $poly(n, m,) \cdot 2^{4(n-m)}\binom{n}{m}(8(n-m))^m$ different records with m pairs placed. Here, $2^{4(n-m)}$ is an overestimation for our possible assignments of Booleans, $\binom{n}{m}$ is the different ways to choose pairs, 2^m is from start-points being in either $S_{1,m}$ or $S_{2,m}$, and $(4(n-m))^m$ comes from the number of ways to place end-points into the available $S_{i,m}$. In total we hence have roughly $\sum_{0=m}^{n} 2^{4(n-m)}\binom{n}{m}(8(n-m))^m$ records, which is linear in terms of the maximum value of $2^{4(n-m)}\binom{n}{m}(8(n-m))^m$. Since $2^{4(n-m)}\binom{n}{m}8^m \leq 32^n$ we are left with having to maximize $(n-m)^m$. Since $n, m, n-m \geq 0$ and $n, m \in \mathbb{N}$, replacing $n-m$ with x (and m with $n-x$) and derivation gives us $\frac{d}{dx}x^{n-x} = x^{n-x-1}(n-x-x\ln x)$. The maximum must then occur when $n = x + x\ln x$, which have no exact finite expression, but can be estimated as $x = \frac{n}{\ln \frac{n}{\ln n}}$. Since $n < \frac{n}{\ln \frac{n}{\ln n}} \ln \frac{n}{\ln \frac{n}{\ln n}}$ this gives us an overestimation of x, and hence we get a valid upper bound on x^{n-x} by also approximating $n - x = n$. Inserting this in the original formula and assuming n is large enough gives us

$$\left(\frac{cn}{\ln(n/\ln n)}\right)^n = \left(\frac{cn}{\ln n - \ln\ln n}\right)^n \leq \left(\frac{2cn}{\ln n}\right)^n.$$

Hence, we have an upper bound on the form $\mathcal{O}^*((\frac{cn}{\ln n})^n)$ where c is some constant, which completes the proof. □

Compared to the previous best known upper bound of $\mathcal{O}^*((1.0615n)^n)$ [6] this is a clear improvement. In fact, we even changed the category ALLEN belongs to by pushing it below $\mathcal{O}^*((cn)^n)$ for every constant $c > 0$. This may be a major cornerstone towards a single-exponential algorithm for ALLEN.

While the algorithm presented here only decides consitency by outputing a *True* or *False* answer, one can using standard implementations of e.g. backpoints also use the algorithm to yield solutions to satisfiable instances. Or even modify the algorithm to instead count solutions.

4. CARDINAL DIRECTION CALCULUS

In this section we focus on CDPA. While CDPA can be reduced to a subproblem of ALLEN, the problem is interesting in its own right and is typically viewed as an independent formalism. The CDPA formalism also allows significantly better complexity than the general case of ALLEN and we will show that CDC can be solved in $O^*((\frac{cn}{\log n})^{2n/3})$ time for a constant c.

We achieve this in three steps: first we modify the algorithm from Eriksson & Lagerkvist [6] to work in the now two-dimensional space. Second, we show that by running the modified version four times in different combinations of directions, while not storing orderings longer than roughly $n/3$, the worst case is significantly improved. Last, we show how the method from Section 3 can be applied, yielding the final complexity of $\mathcal{O}^*((\frac{cn}{\log n})^{2n/3})$.

Briefly, the algorithm presented in Eriksson & Lagerkvist [6] works on intervals by dynamic programming and adding points in a brute force way (regardless of whether they are start-points or end-points) in sets of equality, one set of a time, to the end of the working ordering. While the complexity of this approach is generally bounded by the size of the ordering, we can further bound the size of said ordering, since if we remove whole intervals from the ordering when both start- and end-point are placed, the ordering will never contain any end-points, except in the very last position.

For our first approach to CDPA we follow a similar strategy but in two dimensions: we use dynamic programming and keep track of two active orderings, one in the direction of positive x-relations, and one positive y-relations direction. We also keep track of points already having been placed into both orderings, and as a result we also, directly or indirectly, know which variables that have not been used in either ordering. Effectively, we have records $(S, (S_x, \leq_x), (S_y, \leq_y))$ where $(S_x \cup S_y) \subseteq S \subseteq V$ while $S_x \cap S_y = \emptyset$ and both $(S_x, \leq_x)$ and $(S_x, \leq_x)$ are total orders. By only placing (sets) of points into the smaller of the two orderings, and only at the very end, and then "forgetting" points used in both orderings, we achieve the worst case for this approach. Again, these partitions of S form an ordered partition, but we will not strictly use them as such.

Lemma 5. CDPA *can be solved in* $\mathcal{O}^*(2^{3n/2}OBN(n/2)^2)$ *or* $\mathcal{O}^*((0.7506n)^n)$ *time with dynamic programming.*

Proof. (Sketch) Consider the records we described earlier. Correctness, similar to how it worked for ALLEN, comes from CDPA only containing binary constraints. This allows us to only check constraints once a variable has been fully placed, since we then know how said variable relates to all other variables, and hence know if the constraint is satisfied or not. Once a constraint has been satisfied, we can thereafter forget the variable, since we will not move it inside our ordering, and hence any constraints must keep being satisfied. For the complexity we applied the restriction that we cannot add new variables to the larger of the two orderings, unless they are equal, and we remove variables from the orderings once they are in both. This leads to a situation where we can in the worst case have two orderings of size $n/2$ and 2^n ways to partition the variables between those two, leading to $O^*(2^nOBN(n/2)^2)$ records. Finally, for each worst-case record there are at most $O^*(2^{n/2})$ subsets of V that we can add to the end of the smaller of the two orderings. Hence we do at most $O^*(2^{n/2})$ for each worst-case record. For non-worst-case records, we may do more work (up to $O^*(2^n)$),

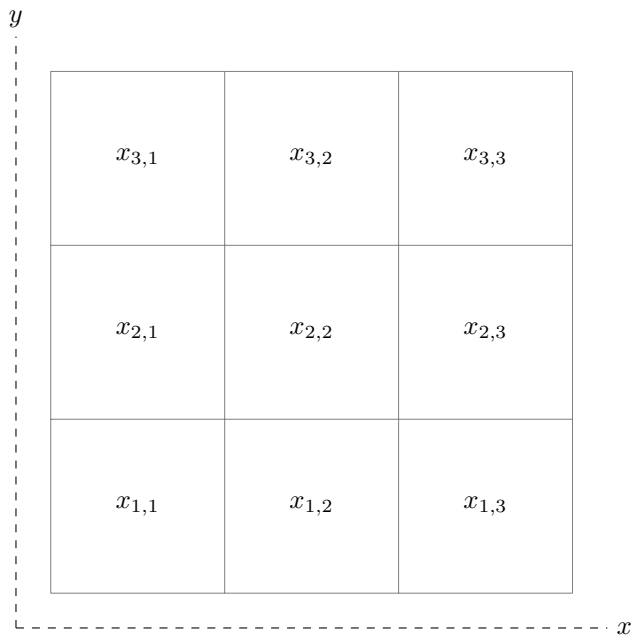

FIGURE 1. Graphic representation of the nine sets described by V, $Sub_{k_{++}}(T_x^+)$, $Sub_{k_{++}}(T_y^+)$, $Sub_{k_{--}}(T_x^-)$ and $Sub_{k_{--}}(T_y^-)$. E.g. $x_{1,1} = |Sub_{k_{++}}(T_x^+) \cap Sub_{k_{++}}(T_y^+)|$, $x_{1,2} = |Sub_{k_{++}}(T_y^+) \setminus Sub_{k_{++}}(T_x^+) \cup Sub_{k_{--}}(T_x^-)|$, $x_{1,3} = |Sub_{k_{++}}(T_y^+) \cap Sub_{k_{--}}(T_x^-)|$ and $x_{2,1} = |Sub_{k_{++}}(T_x^+) \setminus Sub_{k_{++}}(T_y^+) \cup Sub_{k_{--}}(T_y^-)|$.

but as these records are significantly fewer, the asymptotic behavior is dominated by the worst-case ones. Hence, we have the complexity of $O^*(2^{3n/2}OBN(n/2)^2)$ or $O^*((0.7506n)^n)$. □

This is already enough to surpass the previous best result for ALLEN [6] but does not beat the bound from Section 3. To improve this approach for CDPA consider the scenario of having the solution be all points on a diagonal line from positve X and negative Y towards negative X and positive Y. This represents the worst-case for the algorithm outlined previously and naturally raises the question of whether one could order in negative x- or y-direction instead (since this would yield $\mathcal{O}^*(1)$ complexity for this case instead). However, there are of course other cases (we could just mirror this example along either axis) that are instead the worst case for this approach. So, instead we try all four possible combinations of directions: positive x + positive y, positive x + negative y, negative x + positive y and negative x + negative y. Using this approach and not allowing the orderings to grow larger than some maximum, we are able to avoid all these worst cases.

Definition 6. *Given a finite total ordering $(S, <)$ and an integer $k \geq 0$ let $Sub_k((S, <))$ be the first k elements in $(S, <)$, or S if $k \geq |S|$.*

Lemma 7. *For any distribution of a finite set of two dimensional points V in two dimensional space, there are two total orders $T_x = (V, \leq_x)$ and $T_y = (V, \leq_y)$ such that for all $u, v \in V$ if $u_x \odot v_x$ then $u \odot_x v$ and if $u_y \odot v_y$ then $u \odot_y v$ and such that $|Sub_k(T_x) \cap Sub_k(T_y)| \geq k - 1/3n$ for every $0 \leq k \leq n$.*

Proof. Constructing orderings such that such that for all $u, v \in V$ if $u_x \odot v_x$ then $u \odot_x v$ and if $u_y \odot v_y$ then $u \odot_y v$ is trivial. From here on, we assume that n is a multiple of 3. Take the four total orders that can be

generated by sorting our distribution in (1) either positive or negative and (2) either x- or y-direction. Call these four total orders $T_x^+ = (V, \leq_x^+)$, $T_x^- = (V, \leq_x^-)$, $T_y^+ = (V, \leq_y^+)$ and $T_y^- = (V, \leq_y^-)$. Let $k_{++} > n/3$ be the smallest integer such that $|Sub_{k_{++}}(T_x^+) \backslash Sub_{k_{++}}(T_y^+)| = n/3$ but $|Sub_{k_{++}+1}(T_x^+) \backslash Sub_{k_{++}+1}(T_y^+)| > n/3$, or if no such integer exists, then $k_{++} = n$. Similarly we define k_{--} for T_x^- and T_y^-, k_{+-} for T_x^- and T_y^+ and k_{-+} for T_x^+ and T_y^-. By symmetry and renaming we can safely assume that k_{++} is not smaller than any of k_{+-}, k_{-+} and k_{--}. Using $Sub_{k_{++}}(T_x^+)$, $Sub_{k_{--}}(T_x^-)$, $Sub_{k_{++}}(T_y^+)$ and $Sub_{k_{--}}(T_y^-)$ we now partition V into nine subsets and label these as shown in Figure 1. We will now prove that $k_{++} = n$ using proof by contradiction. First we assume that $k_{++} < n$ but that $k_{++} + k_{--} \geq n$:

In this case we know that $n - k_{++} \leq k_{--}$, hence

$$|Sub_{n-k_{++}}(T_x^-) \setminus Sub_{n-k_{++}}(T_y^-)| \leq n/3$$

and that

$$|Sub_{n-k_{++}-1}(T_x^-) \setminus Sub_{n-k_{++}-1}(T_y^-)| \leq n/3.$$

But

$$Sub_{k_{++}}(T_x^+) \setminus Sub_{k_{++}}(T_y^+) = Sub_{n-k_{++}}(T_y^-) \setminus Sub_{n-k_{++}}(T_x^-)$$

and

$$Sub_{k_{++}+1}(T_x^+) \setminus Sub_{k_{++}+1}(T_y^+) = Sub_{n-k_{++}-1}(T_y^-) \setminus Sub_{n-k_{++}-1}(T_x^-).$$

This leads to a contradiction since

$$n/3 \geq Sub_{n-k_{++}-1}(T_y^-) \setminus Sub_{n-k_{++}-1}(T_x^-) > n/3.$$

Hence, our assumption must be incorrect.

In our second case we assume that $k_{++} < n$ but that $k_{++} + k_{--} < n$: Our first observation is that $x_{1,2} + x_{1,3} = x_{2,3} + x_{1,3} = n/3$ by our definition of k_{++}. For simplicity we will assume $x_{1,2} = x_{2,3}$ and similarly $x_{2,1} = x_{3,2}$. Again, by arguments about symmetry we can safely assume that $x_{1,3} \geq x_{3,1}$. We have that $x_{1,3} \geq n/6$, otherwise $x_{1,2} + x_{1,3} + x_{2,3} + x_{2,1} + x_{3,1} + x_{3,2} > n$. By our definition of K_{++}, $x_{1,1} < n/3$ must hold, otherwise $Sub_{k_{++}}(T_x^+) \cup Sub_{k_{++}}(T_y^+) = V$ and $k_{++} = n$. Similarly $x_{1,1} \geq 1$, otherwise $x_{1,2} = x_{2,1} = x_{3,2} = x_{2,3} = n/3$, which can only be true if $n = 0$ and then $k_{++} = n$. Assume that $x_{3,1} = x_{1,3} = n/3$. This implies that $x_{1,1} + x_{2,2} + x_{3,3} = n/3$ and $x_{1,2} = x_{2,1} = 0$ and a situation where $k_{+-} > k_{++}$, as for all $n/3 \leq k \leq n$ then $k - |Sub_k(T_x^+) \cap Sub_k(T_y^-)| \leq n/3$. Hence this also contradicts our original assumption of $k_{++} \geq k_{+-}$. So $x_{3,1} < n/3$ and $x_{2,1} > 1$ as $x_{2,1} + x_{3,1} = n/3$. But this means that $x_{1,1} + x_{1,2} + x_{1,3} + x_{2,3} + x_{3,3} < n/3$ (recall that all $x_{i,i}$ are disjoint) which in turn implies that $Sub_{k_{++}}(T_x^-) \setminus Sub_{k_{++}}(T_y^+) < n/3$ and hence $k_{++} < k_{+-}$. But $k_{++} \geq k_{+-}$ so we again have a contradiction! Hence $k_{++} = n$ is the only viable alternative, completing our proof. □

By Lemma 7 the following result becomes fairly straightforward.

Lemma 8. CDPA *can be solved in* $\mathcal{O}^*((cn)^{2n/3})$ *time with dynamic programming.*

Proof. From Lemma 7 we know that there is a combination of directions such that we at most need to keep track of the exact ordering of $n/3$ points in each direction, yielding a factor $OBN(n/3)^2$. However, Lemma 7 hides a small detail in that the last set of variables added to our total ordering(s) may bring the total ordering above $n/3$ variables. But as this only occurs for the very last part of the ordering, it only adds a factor $2^{O(n)}$. On top of that the variables also needs to be partitioned into which are part of which of the two orderings, which have already been used, and which have not yet been used, giving a factor of $2^{O(n)}$. Combined we hence have a complexity of $O((cn)^{2n/3})$ for some constant c. □

To improve this approach further we implement the same methods as we used in Section 3: instead of strictly using orderings we use records containing pairs (S_i, B_i) where S_i are subsets of V and B_i is the Boolean keeping track of if we can add new variables to the given set or not. Similarly to the ALLEN case we also limit the number of such pairs we allow at once. What differs from ALLEN to CDPA is that we, again, for CDPA need to keep track of two orderings, giving us both $(S_0^x, B_0^x), \ldots, (S_m^x, B_m^x)$ and $(S_0^y, B_0^y), \ldots, (S_m^y, B_m^y)$. Hence the final form of the records are $(S, (S_0^x, B_0^x), \ldots, (S_m^x, B_m^x), (S_0^y, B_0^y), \ldots, (S_m^y, B_m^y))$. Further, the notation of left-heaviness needs to be adapted to the use of two ordered partitions: For all $i \in \{0, \ldots, m\}$ every variable in some S_i^x also needs to be in either S_0^y or S_1^2 and every variable in S_i^y also needs to be in either S_0^x or S_1^x. By also balancing $S_0^x \cup S_1^y$ and $S_0^y \cup S_1^y$ to remain of roughly similar size, we also keep the advantages from our earlier improvements to CDPA. Pair-placing here means that once we place a variable in S_1^x or S_1^y we also simultaneously place it in S_i^y or S_i^x respectively. With this approach, combined with similar logic to our second improvement for CDPA, we at most need as many holes as the current number of unplaced variables. If more is needed, the solution is easier to find in another orientation.

Theorem 9. CDPA *can be solved in* $\mathcal{O}^*((\frac{cn}{\log n})^{2n/3})$ *time using dynamic programming with sublinear partitioning.*

Proof. (Sketch) By using records of the form

$$(S, (S_0^x, B_0^x), \ldots, (S_i^x, B_i^x), (S_0^y, B_0^y), \ldots, (S_j^y, B_j^y)),$$

balancing $S_0^x \cup S_1^y$ and $S_0^y \cup S_1^y$ to roughly equal size, and using our substitute for left-heaviness and pair-placing, we can apply Algorithm 2 with only a few minor modifications. While we in Lemma 8 effectively found the optimum for $|Sub_{k_{++}}(T_x^+) \setminus Sub_{k_{++}}(T_y^+)|$! we are here instead interested in optimizing

$$(2 \cdot |V \setminus \{Sub_{k_{++}}(T_x^+) \cup Sub_{k_{++}}(T_y^+)\}| + 1)^{|Sub_{k_{++}}(T_x^+) \setminus Sub_{k_{++}}(T_y^+)|}.$$

I.e., assuming there are m unplaced variables we need at most m holes in both the x and y direction, while partitioning our current variables into the roughly $2m + 1$ possible sets in each direction, yielding $(2m + 1)^{(1-1/3)n-m}$. Since this function, for large n, has a smaller optimum than $n!$, our complexity is directly given by Lemma 8 combined with the same logic as for Theorem 4. Similarly, correctness also follows via similar arguments to those in Theorem 4. □

With Theorem 9 in place we now have an $o(n)^{2n/3}$ algorithm for CDPA.

5. DISCUSSION AND CONCLUSION

While the results presented in the paper yield significant improvement to the worst-case complexity of ALLEN and CDPA, a plethora of questions and open ends remain.

For ALLEN we consider two major open questions left by our results; the first one being the true value of c and the second that our approach with using left-heaviness seems overly restrictive. Is there a better algorithm that does not necessitate the restriction of left-heaviness and instead allows merging sets to both the left and the right? While removing the restriction of left-heaviness is not difficult, it significantly complicates the analysis, and we did therefore not attempt it.

The questions raised for ALLEN naturally also hold for CDPA. However, in this case the value of the exponent is arguably of more interest. While we saw in Section 4 how the exponent $2n/3$ occurred naturally, many general problems solvable in $2^{\mathcal{O}(n)}$ time often have $2^{\mathcal{O}(\sqrt{n})}$ algorithms for the two-dimensional variants. Could this imply that there might be an $\mathcal{O}^*((cn)^{n/2})$ algorithm for CDPA? Finding such an algorithm, and maybe even improving it further to $\mathcal{O}^*((\frac{cn}{\log n})^{n/2})$ would be very interesting.

The main idea behind the dynamic programming with sublinear partitioning scheme is that we can dynamically restrict the amount of information stored. To the best of our knowledge, this approach has not been explored before in the literature, and we believe that it can be expanded further. For example, while we

here focus on only one type of structure, there is nothing preventing us from storing multiple, e.g. exponentially many, different types of structures simultaneously. This concept would likely be enough to at least match the $2^{\mathcal{O}(n \log \log n)}$ results for ALLEN restricted to only interval length one of [2], while keeping the $\mathcal{O}^*((\frac{cn}{\ln n})^n)$ complexity for general ALLEN.

In conclusion, we have in this paper seen how we can use structural properties required by solutions for ALLEN and CDPA instances to achieve, significantly, improved upper bounds for these problems. In particular we have seen how we using dynamic programming combined with optimizing how much information we store in each iteration, can solve ALLEN in $\mathcal{O}^*((\frac{cn}{\ln n})^n)$ time CDPA in $\mathcal{O}^*((\frac{cn}{\ln n})^{2n/3})$ time.

ACKNOWLEDGEMENTS

The first author is partially supported by the *National Graduate School in Computer Science* (CUGS), Sweden. The second author is partially supported by the Swedish Research Council (VR) under grant 2019-03690.

REFERENCES

[1] J. F. Allen and J. A. G. M. Koomen. Planning using a temporal world model. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-1983)*, pages 741–747. William Kaufmann, 1983.

[2] K. K. Dabrowski, P. Jonsson, S. Ordyniak, and G. Osipov. Fine-grained complexity of temporal problems. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR-2020)*, pages 284–293, 2020.

[3] P. Denis and P. Muller. Predicting globally-coherent temporal structures from texts via endpoint inference and graph decomposition. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pages 1788–1793. IJCAI/AAAI, 2011.

[4] J. Dorn. Dependable reactive event-oriented planning. *Data & Knowledge Engineering*, 16(1):27–49, 1995.

[5] F. Dylla, J. H. Lee, T. Mossakowski, T. Schneider, A. V. Delden, J. V. D. Ven, and D. Wolter. A survey of qualitative spatial and temporal calculi: Algebraic and computational properties. *ACM Computing Surveys (CSUR)*, 50(1):7:1–7:39, Apr. 2017.

[6] L. Eriksson and V. Lagerkvist. Improved algorithms for Allen's interval algebra: a dynamic programming approach. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI-2021)*, pages 1873–1879, 2021.

[7] L. Eriksson and V. Lagerkvist. A multivariate complexity analysis of qualitative reasoning problems. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 1804–1810. International Joint Conferences on Artificial Intelligence Organization, 7 2022.

[8] A. U. Frank. Qualitative spatial reasoning with cardinal directions. In H. Kaindl, editor, *Proceedings of the 7th Austrian Conference on Artificial Intelligence (ÖGAI-91)*, volume 287 of *Informatik-Fachberichte*, pages 157–167. Springer, 1991.

[9] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *Journal of the ACM*, 40(5):1108–1133, 1993.

[10] P. Jonsson and V. Lagerkvist. An initial study of time complexity in infinite-domain constraint satisfaction. *Artificial Intelligence*, 245:115–133, 2017.

[11] P. Jonsson, V. Lagerkvist, and G. Osipov. Acyclic orders, partition schemes and csps: Unified hardness proofs and improved algorithms. *Artificial Intelligence*, 296:103505, 2021.

[12] L. Mudrová and N. Hawes. Task scheduling for mobile robots using interval algebra. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2015)*, pages 383–388. IEEE, 2015.

[13] R. N. Pelavin and J. F. Allen. A model for concurrent actions having temporal extent. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-1987)*, pages 246–250. Morgan Kaufmann, 1987.

[14] F. Song and R. Cohen. The interpretation of temporal relations in narrative. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-1988)*, pages 745–750. AAAI Press / The MIT Press, 1988.

(L. Eriksson) DEP. COMPUTER AND INFORMATION SCIENCE,, LINKÖPINGS UNIVERSITET, SWEDEN
Email address: leif.eriksson@liu.se

(V. Lagerkvist) DEP. COMPUTER AND INFORMATION SCIENCE,, LINKÖPINGS UNIVERSITET, SWEDEN
Email address: victor.lagerkvist@liu.se