Efficient Algorithms for Dynamic Graph-Based Reasoning Systems

Massimo Bono
XXXII cycle, “Ingegneria Informatica ed Automatica” Curriculum
email: mbono@unibs.it
Tutor: Alfonso Emilio Gerevini

UNIVERSITÀ DEGLI STUDI DI BRESCIA

Università degli Studi di Brescia

January 16, 2020

Introduction

Most Artificial Intelligence fields involve some sort of *reasoning*: given some sort of *knowledge* an agent reasons and acts rationally towards a *goal*.

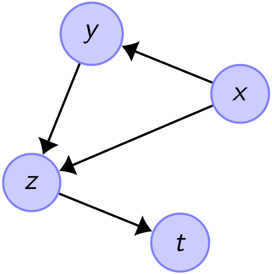
Reasoning for a intelligent artificial agent can be *static* or *dynamic*:

- Static: solve a problem by exploiting some fixed knowledge;
- dynamic: solve a problem with mutable knowledge (e.g., mantaining a property).

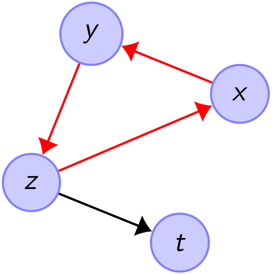
Many applications implicitly involve dynamic reasoning: while such reasoning can be hard, it often is much faster w.r.t. static reasoning.

Introduction (2): static reasoning example

Given a directed graph $G = \langle V_0, E_0 \rangle$, check if G has at least a cycle



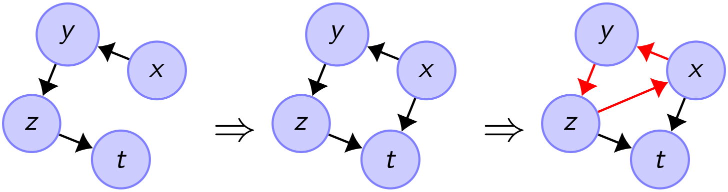
(a) G does not have a cycle



(b) G has a cycle

Introduction (3): dynamic reasoning example

Given a directed acyclic graph $\mathrm{G}_0 = \langle \mathrm{V}_0, \mathrm{E}_0 \rangle$, check if $\mathrm{G}_{i+1} = \langle \mathrm{V}_0, \mathrm{E}_i \cup \{(u_i, v_i)\} \rangle$ $(u_i, v_i \in \mathrm{V}_0, (u_i, v_i) \notin \mathrm{E}_i, i = 0, 1, ..., k)$ has at least a cycle



(a) G_0 does not have a cycle

(b) G_1 does not have a cycle $(u_0 = x, v_0 = t)$

(c) G_2 has a cycle $(u_1 = z, v_1 = x)$.

the dynamic problem can be solved with k runs of the algorithm solving the static problem

however

some knowledge is **not** exploited (i.e., graphs are cumulative).

Dynamic Graph-based Reasoning Systems

The thesis investigates two specific topics involving knowledge represented through graphs:

- **Consistency checking problem in temporal reasoning**: check if a network of constraint encoding temporal information is satisfiable;
- **pathfinding**: given a graph, find the shortest-path from a start vertex to a target.

For both of them, we have considered a dynamic variant of the problem and we propose efficient algorithms for solving such variant. The algorithms have been experimentally evaluated showing significant gains w.r.t. state-of-the-art.

Talk Outline

The talk will be outlined as follows

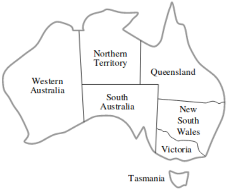
- Decremental Consistency Checking Problem:
  - Background (CSP, consistency, TL-Graph, temporal algebras);
  - motivation and problem definition;
  - Dpasat and Dohsat;
  - experimental results.
- IC-Pathfinding Problem:
  - Background (pathfinding, CPD);
  - motivation and problem definition;
  - CPD-Search;
  - experimental results.
- Conclusion and future works.

Constraint Satisfactory Problem (CSP)

Constraint Satisfactory Problem (CSP)

A CSP Θ consists of a finite set of *variables* $\mathcal{V} = \{x_1, x_2, ...x_n\}$; each variable $x \in \mathcal{V}$ has associated a finite domain of values $Dom(x) = \{v_1, ..., v_k\}$ and a finite set of constraints $\mathcal{C} = \{C_1, ..., C_m\}$.

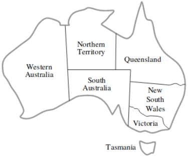
Example: Graph coloring



$$\mathcal{V} = \{WA, NT, SA, Q, NSW, V, T\}$$
$$Dom(x) = \{red, green, blue\}, \forall x \in \mathcal{V}$$
$$\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

Constraint Graph

CSPs can be represented via *constraint graphs*. A *solution* is a complete assignment of the variables which satisfies **all** constraints.



$$\mathcal{V} = \{WA, NT, SA, Q, NSW, V, T\}$$

$$Dom(x) = \{red, green, blue\}, \forall x \in \mathcal{V}$$

$$\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$$

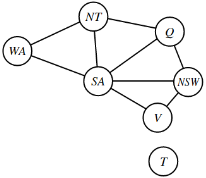


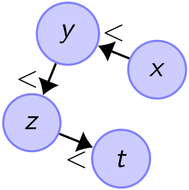
Figure: Constraint graph of Θ.

Figure: Representation of a solution of Θ.

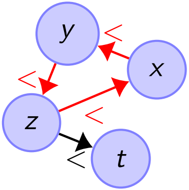
Consistency

Consistency

A CSP Θ is **satisfiable** (**consistent**) iff there is at least one solution (i.e., assignment of values to all the variables $\{x_i = v_i\}$ s.t. no constraint is violated).



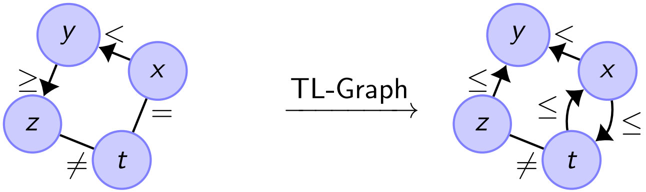
(a) Constraint graph of a consistent CSP



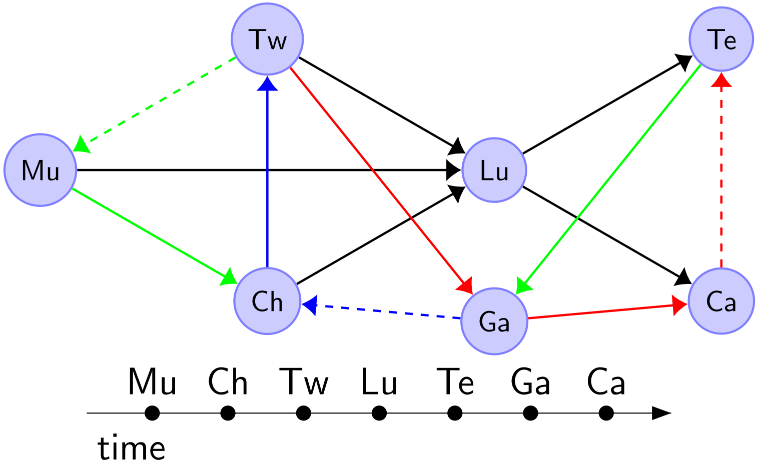
(b) Constraint graph of an inconsistent CSP

Temporal CSPs

- Focus on TCSPs (i.e., CSPs where the variables represent timed events and each constraint involves a relation between 2 events);
- constraints: relations in Point Algebra (PA), Interval Algebra (IA) or its maximal tractable subalgebra, ORD-Horn.
- variables: time points (for PA) or time intervals (for IA);
- IA: $\perp$, 13 base relations and all their possible unions (e.g., two intervals cannot overlap);
- PA: $<, =, >, \leq, \geq, \neq, \top, \perp$ (e.g., $x \neq y$ means that the two events cannot occur at the same time); example of a TCSP over PA:



The Decremental Consistency Checking Problem: Motivation

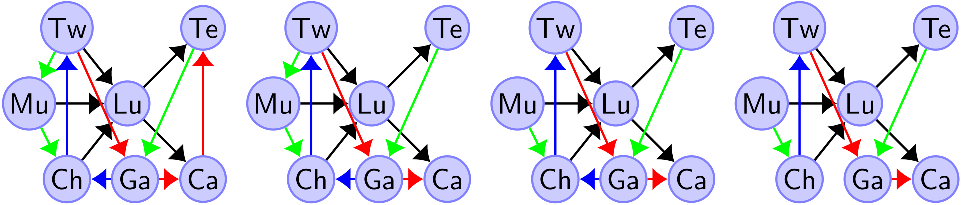


Definition [Bono, Gerevini, 2018]

Given:

- An **inconsistent temporal CSP** Θ over a class of constraints $\mathcal{C}$;
- a **sequence** $\Theta_0, \dots, \Theta_k$ of TCSPs over $\mathcal{C}$ such that $\Theta = \Theta_0$ and Θ_i is obtained from Θ_{i-1} by making one constraint relaxation in Θ_{i-1}, for $i = 1, \dots, k$;

decremental Consistency Checking is the problem of *iteratively deciding the consistency* of every Θ_i starting from Θ_1 until $i = k$ or Θ_i becomes consistent.



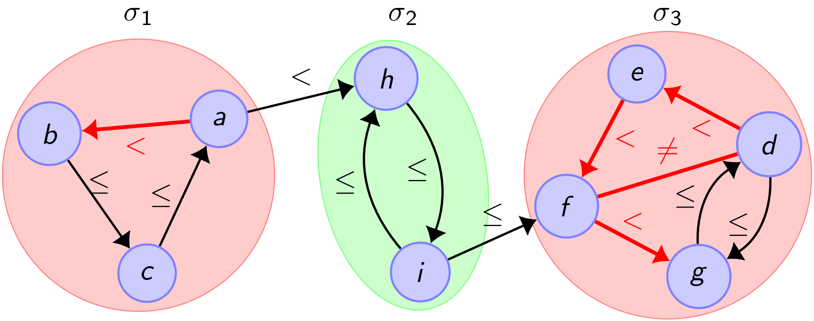
(a) Θ_0 (b) Θ_1 (c) Θ_2 (d) Θ_3

If the TCSPs are all over PA (ORD-Horn), the problem is called D-PSAT (D-OHSAT).

P-SAT [van Beek, 1992]

How to solve the problem of *statically* checking the consistency of a TCSP Θ over PA ($\mathrm{P\text{-}SAT}$)?

- Compute the *Strongly Connected Components* (SCCs) of the TL-Graph of Θ;
- edge labeled either with '$<$' or '$\neq$' is in a SCC subgraph $\Leftrightarrow$ Θ inconsistent.



Solving D-PSAT

Proposed algorithm DPASAT:

- Decremental variant of P-SAT;
- compute useful metadata at the first constraint relaxation allowing us to quickly compute consistency (i.e., problematic SCCs and edges in SCCs with label $\in \{`<', `\neq'\}$);
- mantain such metadata over the sequence of TCSPs.

Solving D-OHSAT

IA algebra is intractable $\Rightarrow$ consider a subalgebra which is tractable (ORD-Horn)

D-OHSAT:

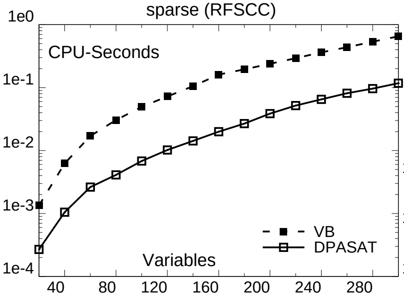
- Each TCSP Ω can be seen as $\pi_1(\Omega) \cup \pi_2(\Omega)$ ($\pi_1(\Omega)$ = TCSP over PA; $\pi_2(\Omega)$: set of at most binary disjunctions where each disjunct is of the form $p\{=, \leq, \neq\}q$ and at least one of them is $p{\neq}q$).

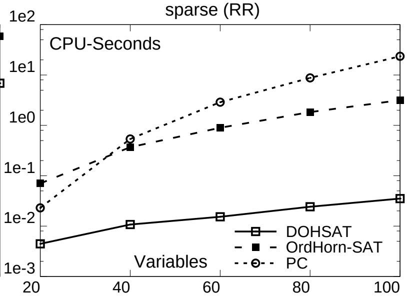
Proposed algorithm DOHSAT:

- Decremental variant of ORD-HORNSAT ([Gerevini, 2005]);
- use DPASAT to manage $\pi_1(\Omega)$;
- compute useful metadata as soon $\pi_1(\Omega)$ becomes consistent allowing us to reuse previous ORD-HORNSAT inferences;
- mantain such metadata over the sequence of TCSPs.

Experimental results

- Each point represents the average CPU-time over 30 instances used by algorithms to solve the decremental problem;
- VB (ORD-HORNSAT and PC) is the state-of-the-art algorithm statically checking consistency of TCSP over PA (ORD-Horn);
- *sparse* graphs are graphs with $\lfloor n \log_2 n \rfloor$ constraints.

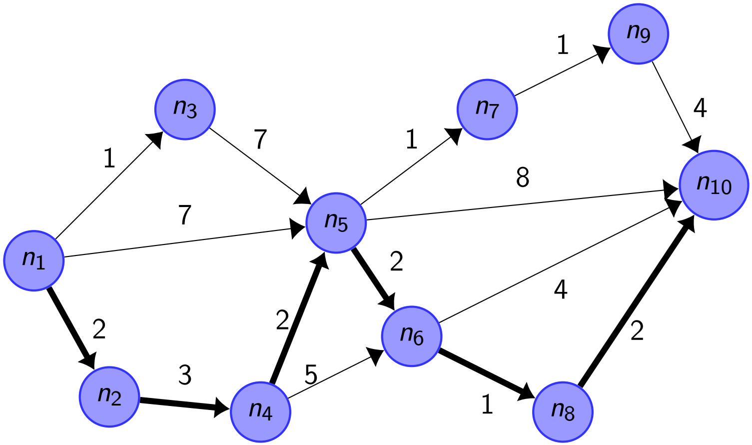




General idea of Pathfinding

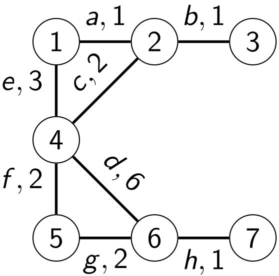
Pathfinding

Given a weighted directed graph $\mathrm{G} = \langle \mathrm{V}, \mathrm{E} \rangle$, $s, t \in \mathrm{V}$, pathfinding is the task of computing a shortest-path from s to t.

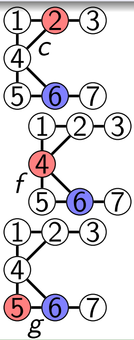


Compressed Path Database (CPD) [Strasser 2014 et al.]

Given a graph, a CPD is a data structure that efficiently stores the first edge of an optimal path from any node s towards any node t.



$(s, t) = (2, 6)$



$CPD[2, 6] = c$

$CPD[4, 6] = f$

$CPD[5, 6] = g$

CPD-Path

Given a graph G and its CPD, source node s and target node t, the $\texttt{CPD-Path}[s, t]$ is the path obtained by concatenating edge $\mathrm{CPD}[s, t]$ with $\texttt{CPD-Path}[\mathit{sink}(\mathrm{CPD}[s, t]), t]$ if $s \neq t$; the empty path otherwise.

A^* [Nilsson et al., 1972]

A^*

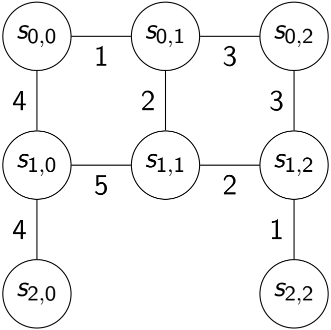
Given a (usually large) weighted directed graph representing a search space, an initial state s and a set of goal states T, the best-first search algorithm A^* computes a path from s to a state in T.

- node evaluation: $f(n) = g(n) + h(n)$;
- $g(n)$: cost of the path for reaching n from s;
- $h(n)$: *estimate* of the cost of the shortest-path for reaching a goal in T from n;
- picks the best node n found up until now, expands its successors and repeat until a goal is found.

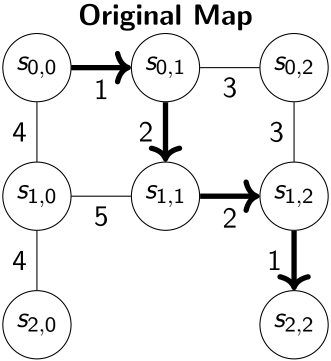
If $h(n)$ is admissible (it never overestimate the actual cost of the shortest path from n to a state in T), then A^* is optimal.

IC-Pathfinding problem

Original Map



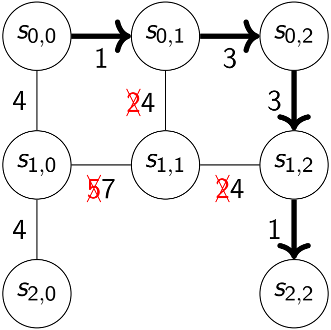
IC-Pathfinding problem



$s_{0,0} \rightarrow s_{0,1} \rightarrow s_{1,1} \rightarrow s_{1,2} \rightarrow s_{2,2}$

IC-Pathfinding problem

Perturbated Map



$s_{0,0} \longrightarrow s_{0,1} \longrightarrow s_{0,2} \longrightarrow s_{1,2} \longrightarrow s_{2,2}$

IC-Pathfinding context

- Path planning episodes are independent;
- each episode has fixed start and target locations;
- graph map is known a priori.

Map edge costs changes (*perturbations*):

- Distribution over map unknown a priori;
- **only increase** original edge costs (e.g., routing in road networks, videogames);
- detected at the beginning of each path planning episode and then assumed fixed.

CPD-SEARCH [Bono, Gerevini, Harabor, Stuckey, 2019]

CPD-SEARCH

A^* variant yielding bounded suboptimal solutions where CPD-Paths are exploited for searching the perturbated map (for brevity, here we focus on the optimal working mode)

s, t : start and target of path finding instance;
n: a search node expanded during CPD-SEARCH;

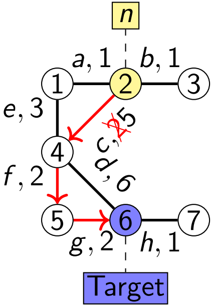
Property

Each node n has implicitly associated, via the CPD, a path to the given target t which is optimal over the original graph (the CPD-Path[n,t]).

- $h_{CPD}[n]$: cost of CPD-Path from n to t using the **original** weights;
- $h'_{CPD}[n]$: cost of CPD-Path from n to t using the **perturbated** weights.

CPD-Paths for deriving an admissible heuristic

- Admissible heuristic: $h_{CPD}[n]$ is a lowerbound of the cost of an optimal path in the perturbated graph (perturbations increase costs).

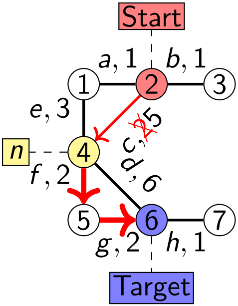


CPD-Paths for early terminating the search

Early search termination: if the `CPD-Path`$[n,t]$ is not perturbated, then we already know an optimal path for going from n to t in the perturbated graph as well!

$$\text{if } h'_{CPD}[n] = h_{CPD}[n]$$
$$\downarrow$$
$$\text{optimal solution is}$$
$$path[s,n] +\!\!+ \texttt{CPD-Path}[n,t];$$

$$(s,t) = (2,6)$$
$$h_{CPD}[4] = 4$$
$$h'_{CPD}[4] = 4$$



CPD-Search

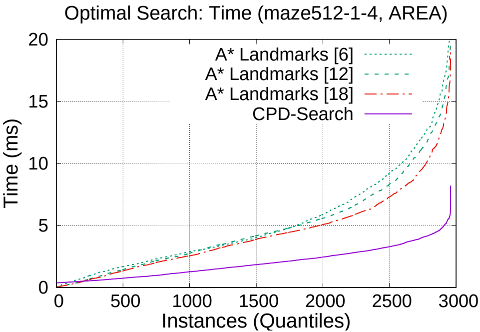
CPD-Search

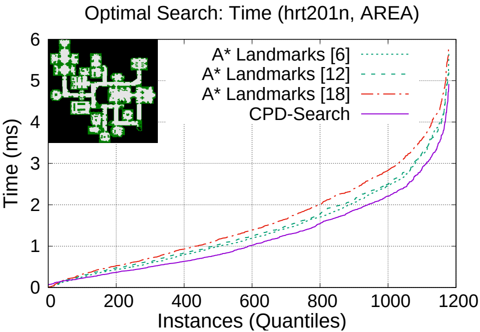
A* variant yielding bounded suboptimal solutions where CPD-Paths are exploited for searching the perturbated map (for brevity, we focus on the optimal working mode)

- CPD-Search maintains the best solution found and solution bounds (allows usage in anytime search);
- bounds computed thanks to $h_{CPD}[n]$ and $h'_{CPD}[n]$;
- a threshold can be set to make the algorithm yields bounded suboptimal solutions (if the threshold is 1, the algorithm yields optimal solutions).

Experiment Results

- Benchmark from moving AI [Sturtevant 2012];
- perturbation policy: along query optimal path we have generated a perturbated area (radius 15, costs increased up to factor of 4);
- comparison performed against ALT[Goldberg and Harrelson, 2005].





Conclusions and Future Works

In the work we have:

- Investigated two new dynamic problems using graph-based knowledge;
- proposed algorithms efficiently solving the problem variants;
- the algorithms (Dpasat, Dohsat, CPD-Search) have been experimentally evaluated. The analysis shows significant gains w.r.t. state-of-the-art;
- **Future works**: The decremental consistency checking problem can be investigated in other algebra (e.g., spatial); or it can be use to find the minimal set of constraint relaxation making an inconsistent TCSP consistent; CPD-Search can be (possibly) further improved by integrating a focal list to improve performance (e.g., when the early termination mechanism is of little help).

Thanks for listening!

Questions???