



- Pro e contro: La CP è potente nella risoluzione di problemi con struttura combinatoria complessa e trova soluzioni rapidamente. Tuttavia, offre poche informazioni strutturali nei problemi complessi, non garantisce direttamente la qualità della soluzione e il processo risolutivo è spesso opaco (black box).

Constraint Programming

Alessandro Hill

rev. 1.0(AH) – 2024



Topics

- Concepts
- Solvers
- Applications



Concepts

La Programmazione a Vincoli ha diverse somiglianze con la Programmazione Matematica, poiché entrambe cercano soluzioni ottimali attraverso variabili e vincoli. Tuttavia, mentre la Programmazione Matematica è orientata principalmente a problemi continui (spesso rappresentati da equazioni e disequazioni), la CP è particolarmente efficace per problemi discreti e combinatori.

1. **Decision variables are represented by domains**
2. **Constraints**
3. **Objective function**

→ Ogni variabile decisionale ha associato un dominio che definisce i valori possibili che quella variabile può assumere.


→ Similarities with Mathematical Programming!

Special cases:

- **SAT Solving**
- **CP over finite domains**

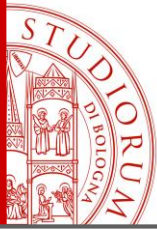
→ SAT Solving: è una sottoclasse di problemi in cui i vincoli sono espressi come formule logiche, e il compito è trovare una configurazione delle variabili che renda la formula vera.

→ CP su domini finiti: è una forma comune di CP dove le variabili possono assumere valori da un insieme finito, rendendo il problema più adatto a essere risolto con algoritmi discreti.



The screenshot shows the homepage of the 'Constraint Solving' website. The header includes a navigation bar with links: Home, Tutorials, Community, Books, Solvers, Conferences, and Forum. Below the header is a large green banner featuring a Rubik's cube and the title 'Constraint Solving'. The banner text explains that constraint programming is a paradigm where relations between variables are stated as constraints, rather than specifying execution steps. Below the banner, there are three main sections: 'Why Constraint Solving?' which quotes E. Freuder, 'Applications of Constraint Solving' which lists real-time applications like manufacturing and telecommunications, and 'About this site' which mentions it is maintained by Martine Ceberio and her.

<http://www.constraintsolving.com>



Domains

Il dominio rappresenta
l'insieme di valori possibili che
una variabile può assumere

- **Domains [of variables] can be Boolean, integer, numerical.**
(commonly finite domain) numeri reali o intervalli continui

Variabili orientate alla pianificazione:

- **Scheduling-oriented variables are provided**
(e.g., Interval variable representing a job/task)

In alcuni casi, specialmente in problemi di pianificazione (scheduling), si utilizzano variabili di tipo "intervallo". Queste variabili rappresentano un'attività o un compito che deve essere eseguito entro un intervallo temporale specifico, facilitando la gestione di task legati al tempo.

Rappresentazione interna dei domini finiti

- **Finite domains are internally represented by intervals.**
(e.g., $x=[1,5]=\{1,2,3,4,5\}$)

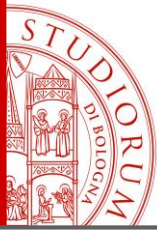
Leads to time-independent models!

Rappresentazione esplicita in presenza di "buchi"

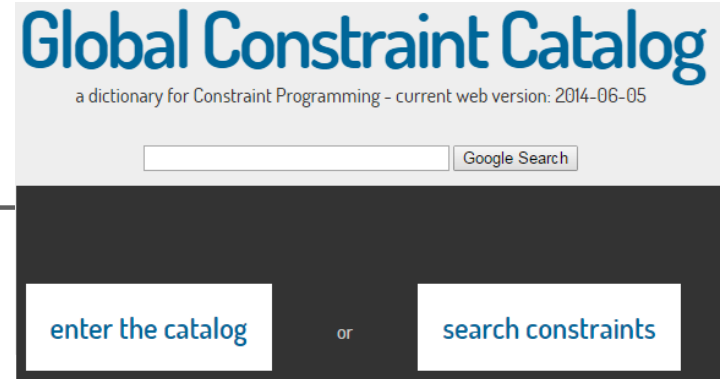
- **Explicit representation when "holes" appear.**
(e.g., $x'=\{1,3,4,10\}$)

Questo approccio semplifica la modellazione e porta a modelli indipendenti dal tempo, in cui le variabili possono essere trattate come insiemi predefiniti.

Quando un dominio contiene valori non consecutivi, viene utilizzata una rappresentazione esplicita. Ad esempio, se il dominio di x' è $\{1, 3, 4, 10\}$, significa che x' può assumere solo questi valori specifici, e non tutti i valori in un intervallo continuo. Questa rappresentazione si usa per garantire che i vincoli siano precisi anche in presenza di domini con valori "salti" o "buchi".



Constraints



I vincoli non devono necessariamente essere lineari; i vincoli possono includere relazioni non lineari e più complesse, rendendo il framework molto flessibile.

I vincoli collegano le variabili (domini)

Constraints link variables (domains) – not necessarily linearly.

Definizione individuale dei vincoli

They can be defined individually...
(e.g., $x^2 > 2$)

Un vincolo può essere definito su una singola variabile o tra variabili specifiche

Vincoli predefiniti

and there are many pre-defined.
(e.g., `ALL_DIFFERENT(x, x', x'')`)

Esistono molti vincoli predefiniti o standard che facilitano la formulazione di problemi comuni. Un esempio è il vincolo `ALL_DIFFERENT`, che applicato a un insieme di variabili impone che tutte assumano valori distinti tra loro.

Propagatore di vincoli

A constraint needs a propagator, which identifies constraint implications on other domains.
(re-apply after domain change)

Un vincolo necessita di un propagatore, un meccanismo che verifica e propaga le implicazioni del vincolo su altre variabili. Quando cambia il dominio di una variabile, il propagatore aggiorna i domini delle variabili correlate per mantenere la consistenza dei vincoli. Questo meccanismo garantisce che, ogni volta che una variabile si modifica, le interdipendenze siano rispettate senza violare alcun vincolo, rendendo l'algoritmo efficiente.

Constraints (cont.)

Questa diapositiva approfondisce l'uso del vincolo ALL_DIFFERENT nella Programmazione a Vincoli, che è uno dei vincoli più comuni e utili, specialmente in problemi di allocazione e assegnazione. Questo vincolo impone che un insieme di variabili assumano valori diversi tra loro.

| | | | | |
|--------------|---|---|---|--------------|
| 1 | 1 | 1 | 1 | 1 |
| 2 | | | | 2 |
| 3 | 3 | 3 | | |
| 4 | | 4 | 4 | 4 |
| 5 | | | | 5 |
| A | B | C | D | E |
| 1 | 1 | 1 | 1 | 1 |
| 2 | | | | 2 |
| 3 | 3 | 3 | | |
| 4 | | 4 | 4 | 4 |
| 5 | | | | 5 |

Uses matching theory

Il vincolo ALL_DIFFERENT(A, B, C, D, E) richiede che ogni variabile nel gruppo assuma un valore unico, senza ripetizioni. In pratica, questo significa che, se A assume il valore 1, nessuna delle altre variabili potrà avere anch'essa il valore 1.

Utilizzando vincoli come ALL_DIFFERENT, possiamo ridurre notevolmente lo spazio di ricerca, poiché si escludono in anticipo combinazioni di valori non valide, rendendo la soluzione del problema più efficiente. Questo vincolo è particolarmente utile in problemi di pianificazione, sudoku, e assegnazione di risorse dove ogni risorsa o slot deve essere assegnato a un'entità unica.

ALL_DIFFERENT(A,B,C,D,E)

Come funziona l'algoritmo (matching theory)?

L'algoritmo che applica questo vincolo utilizza principi di teoria del matching per propagare le restrizioni tra i domini delle variabili. Vediamo come funziona nella figura:

- Domini iniziali: In alto vediamo i domini iniziali di ogni variabile
- Propagazione dei vincoli: Il vincolo ALL_DIFFERENT comincia a ridurre i domini delle variabili per assicurarsi che non ci siano sovrapposizioni nei valori. Questo porta a una propagazione iterativa, che elimina i valori incompatibili dai domini delle altre variabili.
- Domini ridotti: Dopo l'applicazione del vincolo, alcuni valori vengono esclusi dai domini (indicati con le "X" in basso), riducendo le possibili scelte per ciascuna variabile in modo da rispettare il vincolo ALL_DIFFERENT.

Objectives

In Programmazione a Vincoli, è possibile costruire espressioni complesse utilizzando funzioni e operatori come MAX, MIN, IF/THEN, e POWER. Questo significa che è possibile creare espressioni condizionali, espressioni di confronto, e altre funzioni matematiche per modellare problemi complessi. Questa flessibilità permette di rappresentare una vasta gamma di problemi e di vincoli, fornendo un controllo dettagliato sui valori che le variabili possono assumere.

Build any expression

(e.g., use MAX/MIN, IF/THEN, POWER)

Accedere alle proprietà delle variabili intervallo

- **Access interval variable properties**

Serve cautela con le espressioni

- **Some carefulness with expressions needed**

La costruzione di espressioni complesse richiede attenzione per evitare errori e per garantire che i calcoli siano efficienti. Espressioni complesse possono infatti rallentare la risoluzione del problema se non sono formulate in modo ottimale.

Ad esempio, l'uso di espressioni condizionali complesse o calcoli ripetuti può portare a tempi di calcolo elevati, quindi è importante semplificare le espressioni quando possibile o evitare condizioni ridondanti.






Inoltre, alcuni operatori potrebbero comportarsi in modo diverso in base ai valori o ai vincoli presenti, quindi è importante conoscere l'effetto delle espressioni sul modello complessivo.

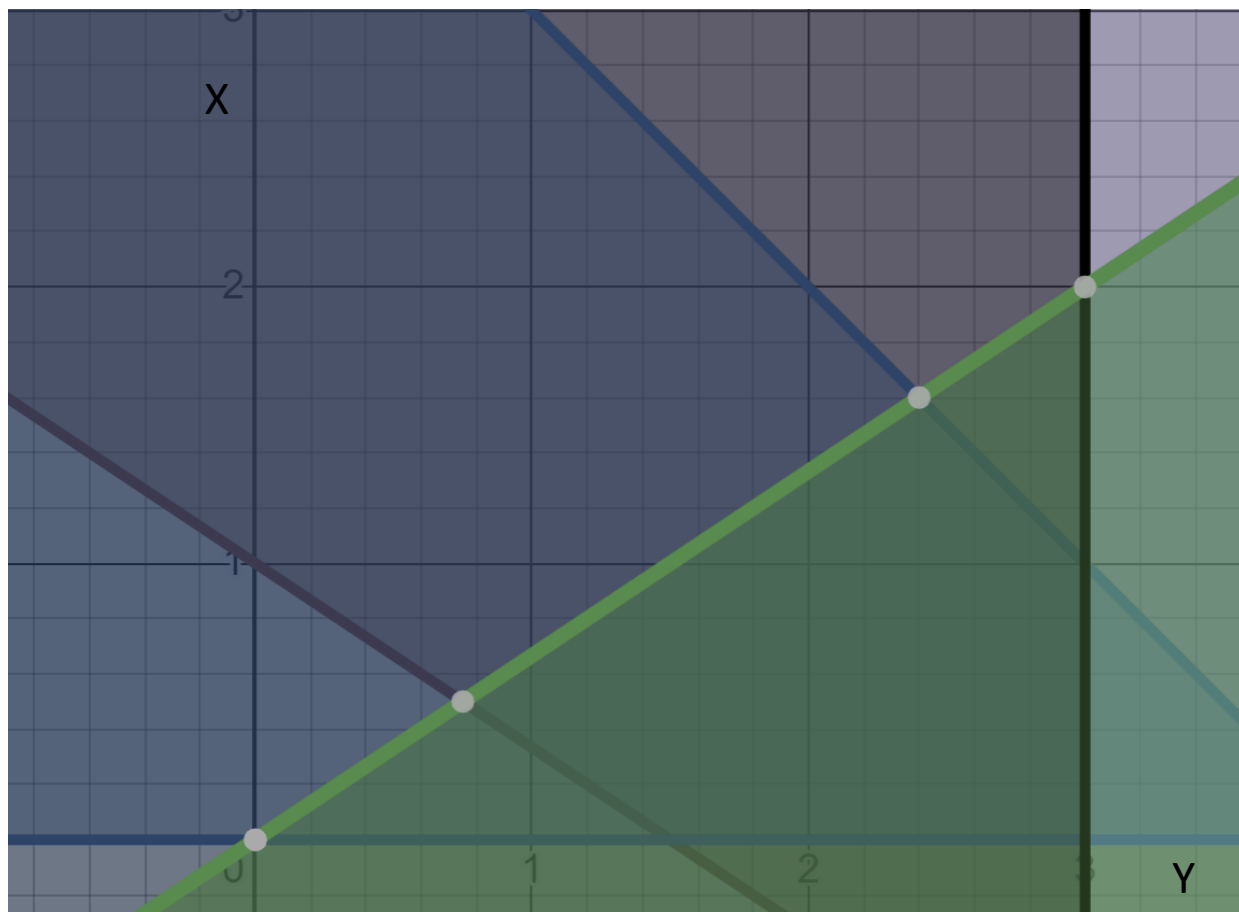
Le variabili intervallo sono un tipo speciale di variabili che rappresentano intervalli di tempo o durate. È possibile accedere a proprietà specifiche delle variabili intervallo, come il tempo di inizio, tempo di fine, e durata dell'intervallo. Questo consente di definire vincoli basati su queste proprietà

Branch & Bound

Il Branch and Bound è un algoritmo fondamentale per risolvere problemi di ottimizzazione combinatoria, in particolare quelli di Programmazione Lineare Intera e Programmazione a Vincoli.

$$\begin{array}{ll}
 \text{Min} & 2X + Y \\
 \text{s.t.} & 2X + 3Y \geq 3 \\
 & X + Y \leq 4 \\
 & 2X - 3Y \geq 0 \\
 & X \leq 3 \\
 & X, Y \geq 0 \\
 & X, Y \text{ intere}
 \end{array}$$

- 1  $2x + 3y \geq 3$
- 2  $x + y \leq 4$
- 3  $2x - 3y \geq 0$
- 4  $y \geq 0$
- 5  $x \leq 3$



Modelling languages

FlatZinc: Standard low-level modelling language

MiniZinc: High-level modelling language

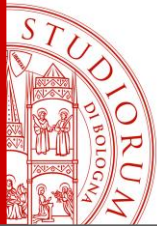
```
% Colouring Australia using nc colours
int: nc = 3;

var 1..nc: wa;   var 1..nc: nt;   var 1..nc: sa;   var 1..nc: q;
var 1..nc: nsw;  var 1..nc: v;    var 1..nc: t;

constraint wa != nt;
constraint wa != sa;
constraint nt != sa;
constraint nt != q;
constraint sa != q;
constraint sa != nsw;
constraint sa != v;
constraint q != nsw;
constraint nsw != v;
solve satisfy;

output ["wa=", show(wa), "\t nt=", show(nt),
        "\t sa=", show(sa), "\n", "q=", show(q),
        "\t nsw=", show(nsw), "\t v=", show(v), "\n",
        "t=", show(t), "\n"];
```





Solvers

Questa slide presenta una panoramica dei solver utilizzati nella Programmazione a Vincoli (Constraint Programming, CP) e le tecniche chiave che essi impiegano per risolvere problemi complessi.

Questa tecnica consiste nel ridurre i domini delle variabili eliminando valori che violerebbero i vincoli, limitando così lo spazio di ricerca. La propagazione dei vincoli si attiva ogni volta che il dominio di una variabile si modifica, influenzando i domini delle variabili collegate. Il filtraggio dei domini aiuta a restringere le possibilità, riducendo il numero di combinazioni da esplorare e rendendo la ricerca più efficiente.

Key ingredients:

- Branch and bound
- Domain filtering (Propagation)

Also:

- 1 • Impact-based branching rules
- 2 • No-good learning
- 3 • Automated tuning
- 4 • Linear programming
- 5 • Neighborhood search
- 6 • Machine learning
- 7 • Probing

Competition: [MiniZinc Challenges](#)
(yearly, 2008-today)

Commercial:
IBM ILOG CP Optimizer
(research: free)

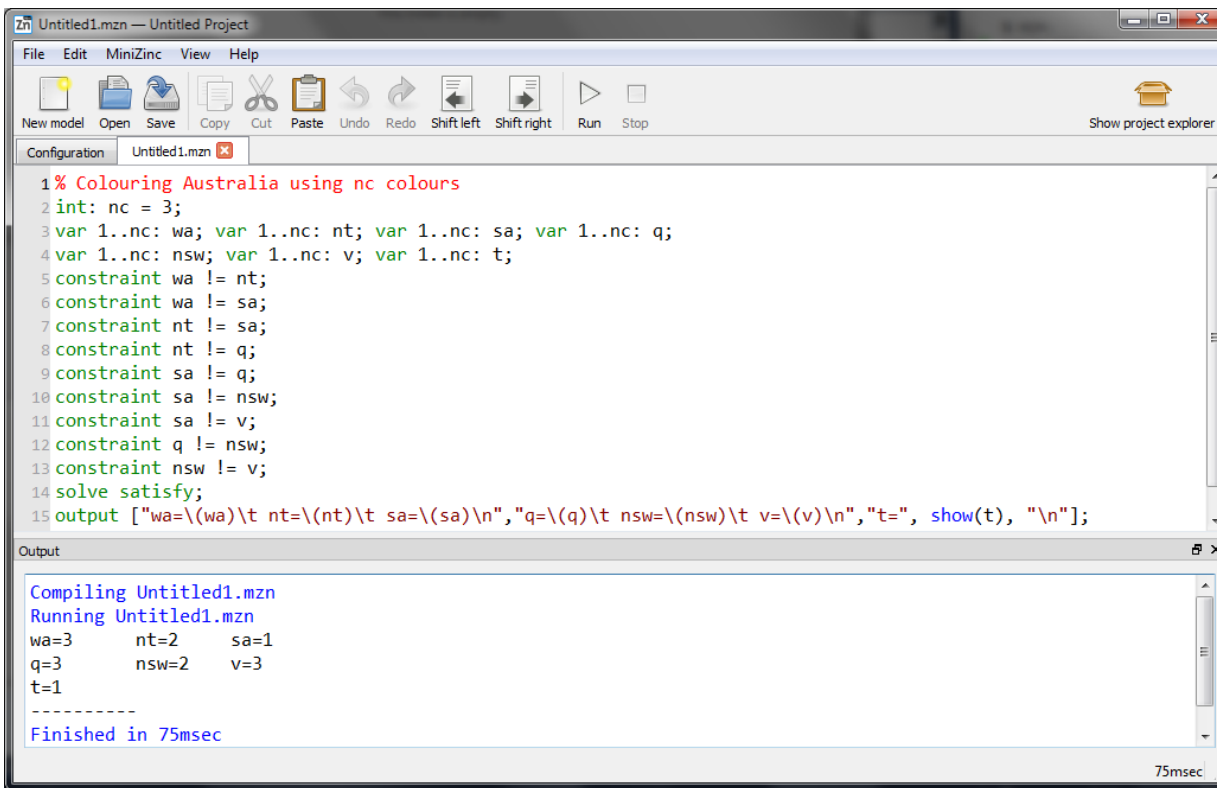
Free:
GECODE (www.gecode.org)
CHUFFED (SAT-hybrid)
CHOCO (java, open-source,
www.choco-solver.org)

...

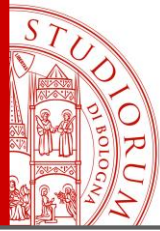
- 1) Tecniche che decidono quale variabile esplorare in base all'impatto che ha sulla riduzione del dominio e sul progresso della ricerca.
- 2) Si tratta di una tecnica che memorizza configurazioni "no-good" (non valide) per evitare di esplorare soluzioni che sono già state scartate come non valide.
- 3) I solver possono adattare automaticamente i parametri di risoluzione in base alle caratteristiche specifiche del problema
- 4) Alcuni solver integrano metodi di Programmazione Lineare (LP) per risolvere sottoproblemi che contengono vincoli lineari
- 5) Una tecnica che cerca soluzioni vicine a una soluzione corrente
- 6) Tecniche di apprendimento automatico vengono talvolta impiegate per migliorare l'efficienza della risoluzione
- 7) Il probing esplora i possibili effetti di decisioni su variabili specifiche per vedere come influenzano il problema complessivo



MiniZinc Tool



- User interface
- Provides modelling language
- Translates model into FlatZinc
- Interfaces various solvers
- Free



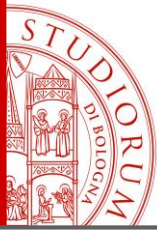
IBM ILOG CP Optimizer

- **Interfaces:** C/C++, Java, Python, OP (decent APIs)
- **Focus on**
 - **Integer models** (allows floating point)
 - **Scheduling**
- **Rich constraint library** (e.g., packing, assignments, resources, precedences, set-up costs/times)
- **Automated tuning**
- **Conflict analysis**

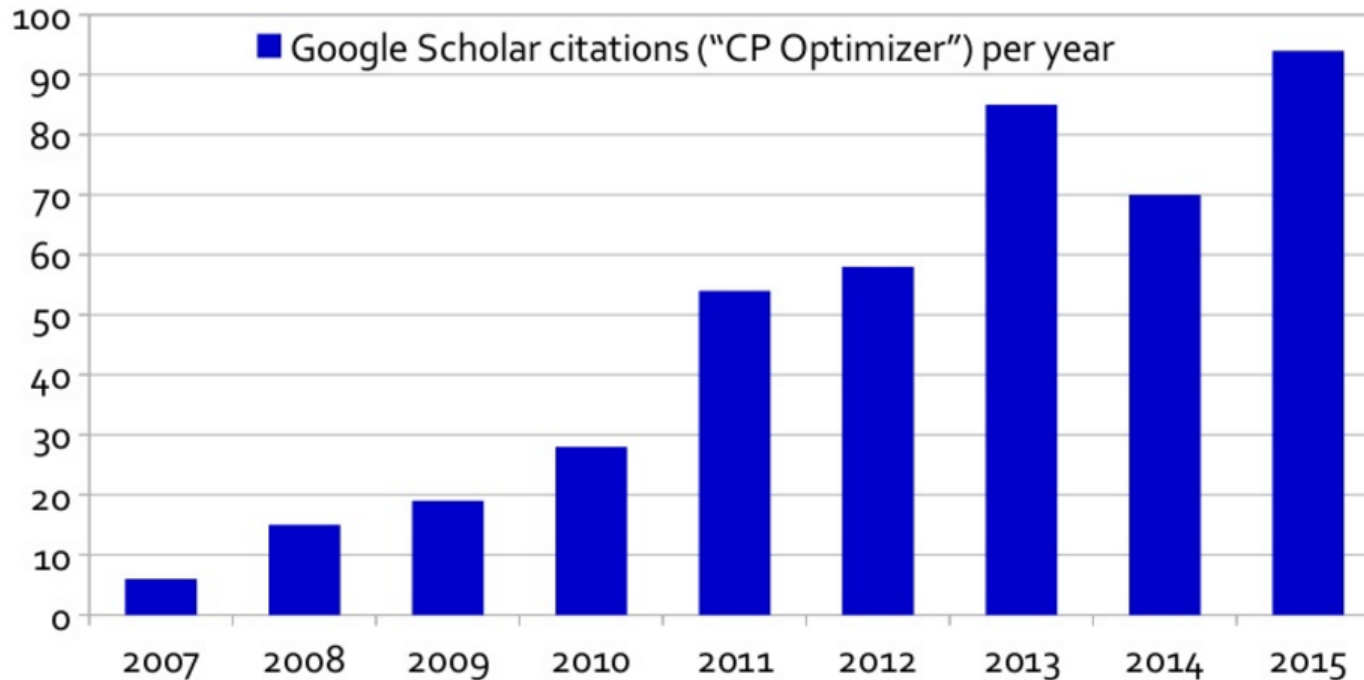
CP Optimizer model for RCPSP:

```
dvar interval a[i in Tasks] size i.pt;  
  
cumulFunction usage[r in Resources] =  
    sum (i in Tasks: i.qty[r]>0) pulse(a[i], i.qty[r]);  
  
minimize max(i in Tasks) endOf(a[i]);  
subject to {  
    forall (r in Resources)  
        usage[r] <= Capacity[r];  
    forall (i in Tasks, j in i.succs)  
        endBeforeStart(a[i], a[j]);  
}
```

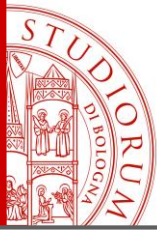
IBM



IBM ILOG CP Optimizer (cont.)



© 2016 IBM Corporation



Successful applications

Questa slide illustra alcune delle applicazioni di successo della Programmazione a Vincoli (Constraint Programming, CP) in vari settori, evidenziando come questo approccio possa risolvere problemi complessi in modo efficace.

- **Scheduling** → Il scheduling è il processo di pianificazione di attività o risorse in un arco di tempo, rispettando vincoli specifici.
- **Timetabling** → Il timetabling riguarda la creazione di orari rispettando i vincoli di disponibilità di risorse
- **Product configuration** → La configurazione di prodotto implica selezionare e combinare componenti di un prodotto rispettando le specifiche richieste dal cliente e i vincoli tecnici.
- ...

Air traffic management: gestione del traffico aereo, un problema complesso che richiede di garantire la sicurezza e la puntualità dei voli.
Semiconductor testing: pianificazione e testing nel processo di produzione di semiconduttori, dove è essenziale coordinare le risorse di testing.
Aircraft assembly: organizzazione dell'assemblaggio degli aeroplani, una serie di attività sequenziali e altamente vincolate.
Flow-shop con buffer: tipo di problema di scheduling in un ambiente produttivo dove i buffer (magazzini temporanei) sono utilizzati per ridurre il tempo di attesa tra le fasi.

MRD: una misura di quanto la soluzione ottenuta con CP si avvicina alla soluzione ottimale. Valori negativi indicano miglioramenti rispetto a benchmark precedenti.

Imp. UBs / # Instances: indica quante soluzioni ottimali sono state migliorate e il numero totale di istanze del problema risolte.

| Bench index | Problem type | MRD | # Imp. UBs / # Instances |
|-------------|-------------------------|--------|--------------------------|
| 1 | Trolley | -10.2% | 15/15 |
| 2 | Hybrid flow-shop | -11.3% | 19/20 |
| 3 | Job-shop w/ E/T | -6.2% | 41/48 |
| 4 | Air traffic management | -7.0% | 1/1 |
| 5 | Max. quality RCPSP | -2.7% | NA/3600 |
| 6 | Flow-shop w/ E/T | -1.1% | 5/12 |
| 7 | RCPSP w/ E/T | -2.1% | 16/60 |
| 8 | Cumulative job-shop | -0.1% | 15/86 |
| 9 | Semiconductor testing | -0.3% | 7/18 |
| 10 | Single proc. tardiness | 0.3% | 0/20 |
| 11 | Open-shop | 0.3% | 0/28 |
| 12 | MaScLib single machine | 0.6% | 0/60 |
| 13 | Shop w/ setup times | 0.4% | 3/15 |
| 14 | RCPSP | 1.2% | 2/600 |
| 15 | Air land | 0.0% | 0/8 |
| 16 | Parallel machine w/ E/T | 1.6% | 4/52 |
| 17 | Job-shop | 1.9% | 0/33 |
| 18 | Flow-shop | 0.9% | 4/22 |
| 19 | Flow-shop w/ buffers | 3.9% | 11/30 |
| 20 | Single machine w/ E/T | 7.4% | 0/40 |
| 21 | Aircraft assembly | 8.7% | 0/1 |
| 22 | Common due-date | 6.8% | 4/20 |

CP: Pros

- Impostazione facile del modello di ottimizzazione

▪ **Easy optimization model setup** → **Model & Run**

Definire il modello e farlo eseguire ("Model & Run"), senza bisogno di specificare algoritmi complessi.

La CP consente una configurazione semplice e rapida dei modelli di ottimizzazione.
- Modellazione di vincoli complessi

▪ **Modeling of complex constraints** →

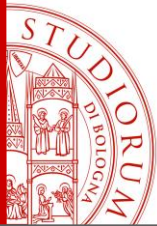
Uno dei punti di forza della CP è la capacità di gestire vincoli complessi, inclusi vincoli logici, vincoli non lineari e vincoli personalizzati.
- **Scheduling-oriented** La CP è particolarmente adatta ai problemi di scheduling, cioè alla pianificazione delle attività nel tempo rispettando risorse limitate e vincoli specifici.
- **Various strong solvers available** Esistono molti solver di CP robusti e ben sviluppati, sia commerciali che open-source
- Efficiente per problemi con struttura combinatoria complessa

▪ **Efficient for problems with complex combinatorial structure (up to a certain size)** →

La CP è particolarmente efficace per risolvere problemi con una struttura combinatoria complessa (problema coinvolge molte combinazioni di scelte discrete) fino a una certa dimensione. Questo include problemi con molti vincoli e variabili discrete.
- **Finds feasible solutions fast** →

La CP è progettata per trovare rapidamente soluzioni che soddisfano tutti i vincoli, anche se non sono ottimali.
- **Detects infeasibility fast**

Se un problema non ha soluzioni che soddisfano tutti i vincoli, la CP può rilevare rapidamente l'infattibilità.



CP: Cons

Nei problemi di grandi dimensioni e particolarmente complessi, la CP spesso fornisce poche informazioni strutturali sulla soluzione o sulla struttura del problema (cioè in CP non abbiamo una rappresentazione geometrica o algebrica chiara dell'interazione tra i vincoli.). Questo significa che, a differenza di altri metodi, in CP è più difficile analizzare il problema e trarre informazioni utili su come ottimizzare o ridurre il problema. Inoltre, senza informazioni strutturali, è difficile decomporre il problema in sottoproblemi più gestibili. In altre parole, diventa complicato suddividere il problema in blocchi indipendenti o semi-indipendenti che possono essere risolti separatamente.

- **Only little structural information for hard large-scale problems**

- **No direct solution quality guarantee**

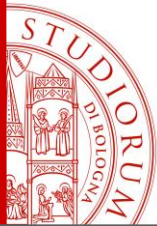
La CP non garantisce necessariamente la qualità della soluzione trovata, a meno che non si raggiunga l'ottimalità completa. In alcuni casi, i solver di CP trovano soluzioni fattibili ma non è garantito che siano ottimali.

- **Black box behavior**

Molti solver di CP funzionano come una "scatola nera" (black box), il che significa che l'utente definisce il problema e il solver lo risolve senza rendere trasparente il processo.

- **Efficient problem formulation is difficult**

Formulare un problema in modo efficiente in CP può essere complesso, poiché richiede una buona conoscenza dei vincoli e delle variabili. Piccole modifiche nella formulazione possono influenzare drasticamente i tempi di risoluzione, quindi trovare una formulazione ottimale richiede spesso esperienza e tentativi. Inoltre, la CP è molto sensibile alla definizione dei vincoli



Some resources

- **IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems**, Philippe Laborie, 2009
- **CP Optimizer Walkthrough**, Paul Shaw, 2013
- **Modeling and Solving Scheduling Problems with CP Optimizer**, Philippe Laborie, 2014
- **An Introduction to CP Optimizer**, Philippe Laborie, 2016
- **Solver challenge:** www.minizinc.org/challenge.html
- **Global Constraint Catalogue:** <http://sofdem.github.io/gccat>
- www.constraintsolving.com
- **MiniZinc tool:** www.minizinc.org