A Hybrid Solver for Large Neighborhood Search: Mixing Gecode and EasyLocal++

Raffaele Cipriano[1] Luca Di Gaspero[2]
Agostino Dovier[1]

1) DIMI - Dip. di Matematica e Informatica
Università di Udine, via delle Scienze 206, I-33100, Udine, Italy

2) DIEGM - Dip. di Ingegneria Elettrica, Gestionale e Meccanica
Università di Udine, via delle Scienze 208, I-33100, Udine, Italy

11 dicembre 2009

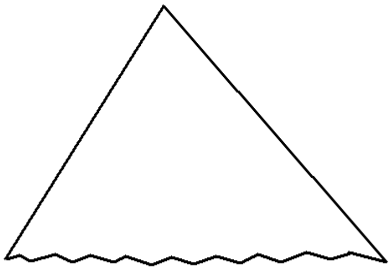
Index

- A declarative programming methodology, parametric on the constraint domain (typically, *finite domains*)
- A *Constraint Satisfaction Problem (CSP) P* is encoded using
  - set of *variables* $X = \{x_1, \ldots, x_k\}$
  - set of *domains* $D = \{D_1, \ldots, D_k\}$
  - set of *constraints* $\mathcal{C}$ over $dom = D_1 \times \cdots \times D_k$
- *Solution*: a tuple $d = \langle d_1, \ldots, d_k \rangle \in dom$ that satisfies every constraint $C \in \mathcal{C}$
- A *Constraint Optimization Problem (COP)* is a CSP with an objective function $f : sol(P) \rightarrow \mathbb{N}$ or $\mathbb{R}$ to be minimized (maximized).

Constraint Programming

Search

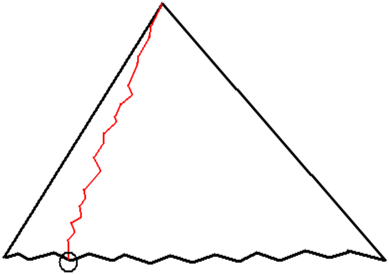
- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.



Constraint Programming

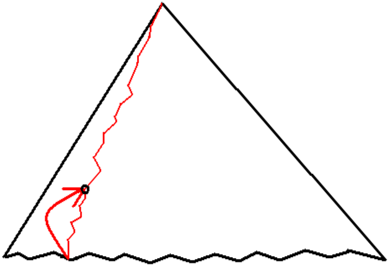
Search

- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.



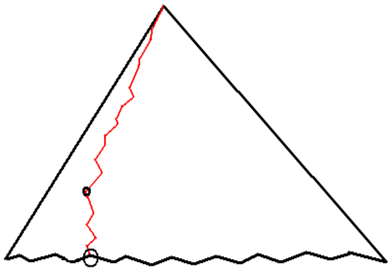
Constraint Programming
Search

- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.



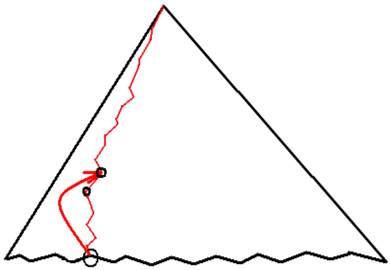
Constraint Programming
Search

- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.

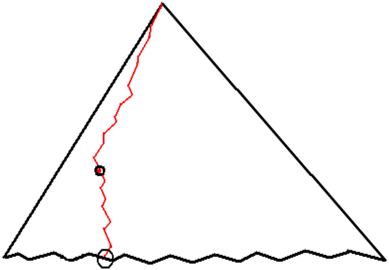


Constraint Programming
Search

- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.



- Alternates of *deterministic phases* (constraint propagation) and *non-deterministic phases* (variable assignment)
- In principle, it is a *Complete* method: the entire search tree is explored trough (intelligent) backtracking
- In practice, *timeouts* are added.



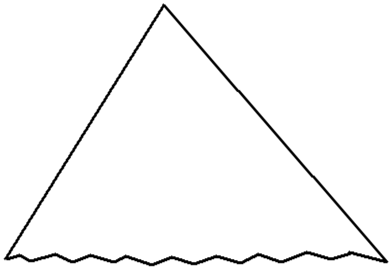
A Hybrid Solver for Large Neighborhood Search
Cipriano, Di Gaspero, Dovier
Preliminary Concepts
Constraint Programming
Local Search
A Hybrid Solver for LNS
General idea
GELATO
A simple experiment
Asymmetric Travel Salesman Problem
Experiments
Conclusions

Local Search

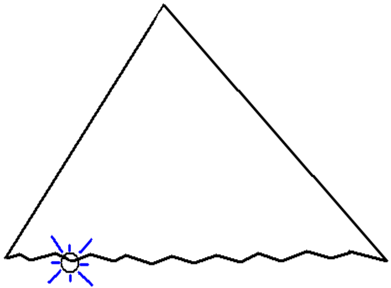
Basics

- A family of heuristic and meta-heuristic methods, based on the idea of *neighborhood*
- A *Constraint Satisfaction/Optimization* problem P is encoded defining
  - the *Search space* S: set of solution; $s \in S$ is *feasible* iff it fulfills the constraints of P
  - the *Neighborhood function*: $\mathcal{N} : S \to \mathcal{P}(S)$; $s' \in \mathcal{N}(s)$ is called a *neighbor* of s
  - the *Objective function*: $f : S \to \mathbb{N}$ or $\mathbb{R}$
- For *Constraint Satisfaction Problems*, we define f as a *distance to feasibility*: number of constraints violated (possibly with different weights)

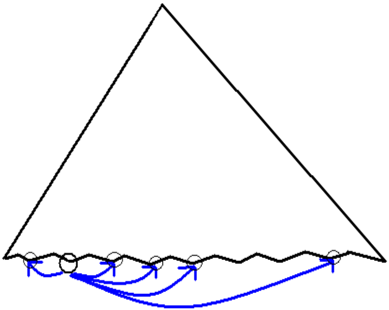
- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



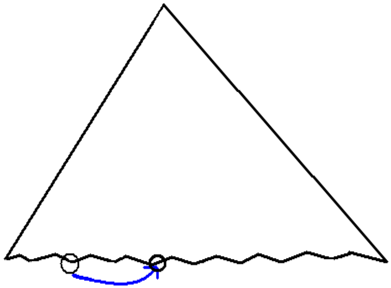
- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



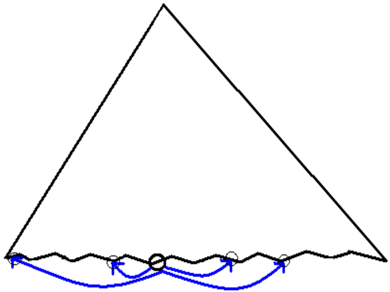
- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



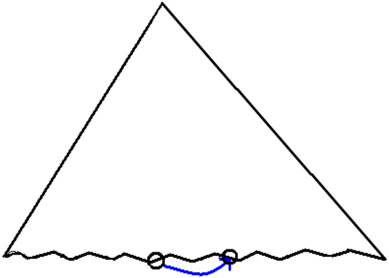
- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



- Starts from an initial $s_0 \in S$, and navigates the search space moving *through the neighborhoods*
- Efficiency:concentrating on some parts of the search space, it can approximate optimal solutions in shorter time
- Incomplete method: it jumps through the solutions



Combining the techniques:
Large Neighborhood Search (LNS)

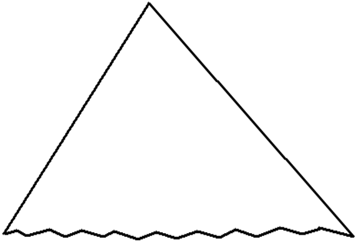
- LNS can be seen as a LS algorithm with a a particular definition of the *neighborhood relation* and of the *neighborhood exploration* strategy
- Instead of small changes to few variables (as usual),
- A "large" subset *FV* of constrained variables (*free variables*) is selected and searched for improving solutions
- Crucial parameters in LNS definition:
  1. *which* variables are in the set *FV* (neighborhood relation)
  2. *how many* variables are in the set *FV* (neighborhood relation)
  3. *how to perform the exploration* on *FV* (neighborhood exploration)
  4. *when stop the exploration* on *FV* (neighborhood exploration)

Combining the techniques:
Large Neighborhood Search (LNS)

- LNS can be seen as a LS algorithm with a a particular definition of the *neighborhood relation* and of the *neighborhood exploration* strategy
- Instead of small changes to few variables (as usual),
- A "large" subset *FV* of constrained variables (*free variables*) is selected and searched for improving solutions
- Crucial parameters in LNS definition:
  1. *which* variables are in the set *FV* (neighborhood relation)
  2. *how many* variables are in the set *FV* (neighborhood relation)
  3. *how to perform the exploration* on *FV* (neighborhood exploration)
  4. *when stop the exploration* on *FV* (neighborhood exploration)

A Hybrid Solver for LNS
General idea

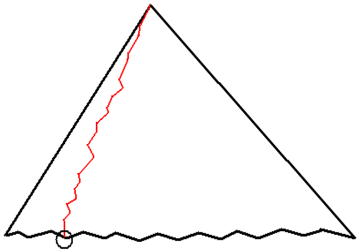
- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



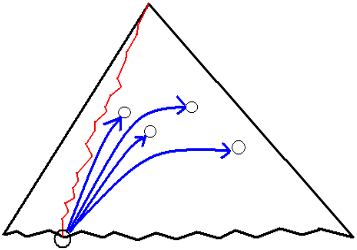
A Hybrid Solver for LNS

General idea

- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached

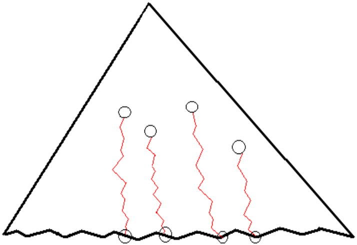


- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



A Hybrid Solver for LNS
General idea

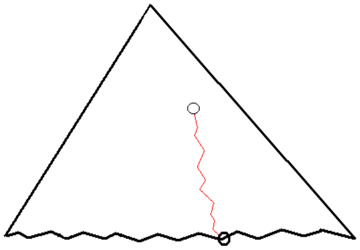
- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



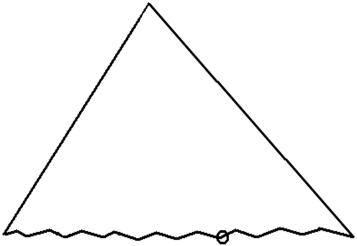
A Hybrid Solver for LNS

General idea

- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



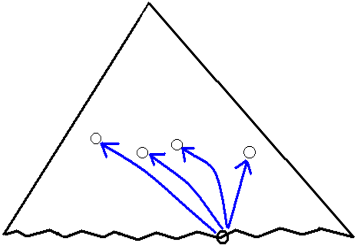
- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



A Hybrid Solver for LNS

General idea

- Inherit both the *flexibility* of CP and the *efficiency* of LS, *alternating CP and LS phases:*
  1. Start from an initial s_0 solution obtained by CP
  2. Selects one or more Large Neighborhoods (sets of *free variables*)
  3. Explore these *Large Neighborhoods* using a CP model
  4. Iterate 2-3 until optimum found or timeout reached



A Hybrid Solver for LNS

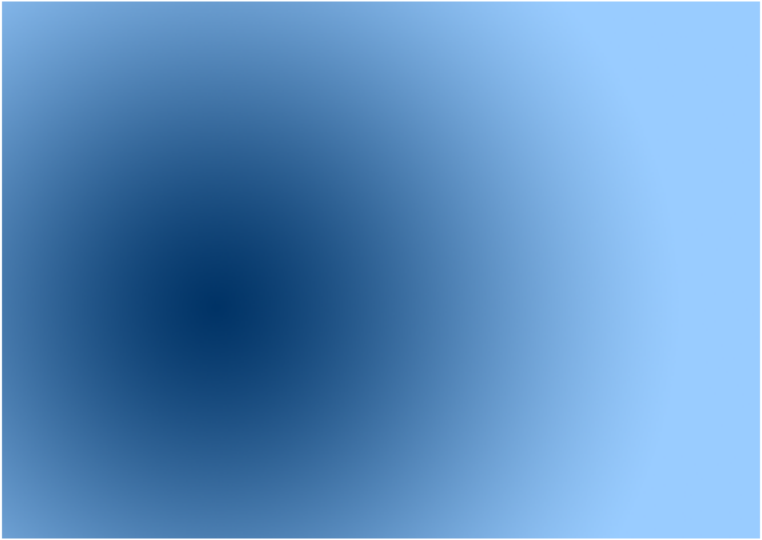
Constraint solver used: GECODE

- Developed by Christian Schulte and Mikael Lagerkvist (KTH, Sweden) and by Guido Tack (Saarland Univ., Germany)
- C++ constraint developing environment
- Stable, strong, complete, growing
- Low-lovel modeling, high efficiency
- Free and open source

- Developed by Luca Di Gaspero and Andrea Schaerf (Univ. of Udine)
- C++ object-oriented framework for LS algorithms design and implementation
- Essential features of most local search metaheuristics, and their possible compositions
- The framework can easily be customized and the code reused
- Free and open source

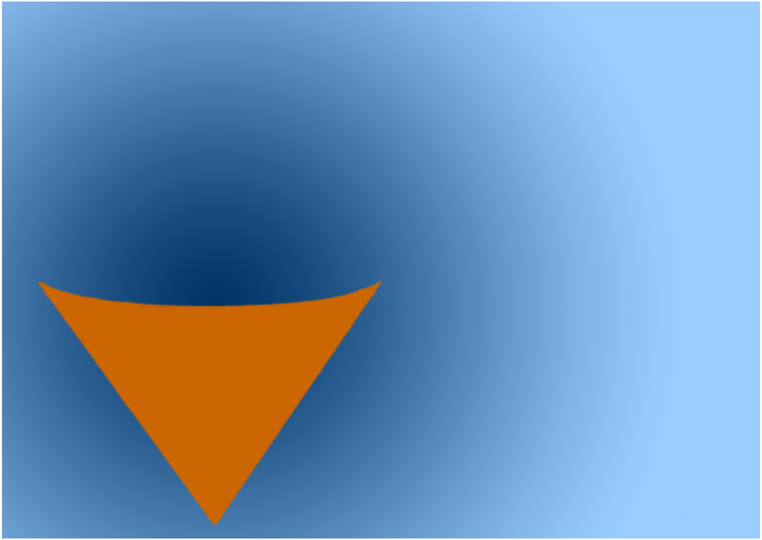
GELATO

IML diagram of the main components



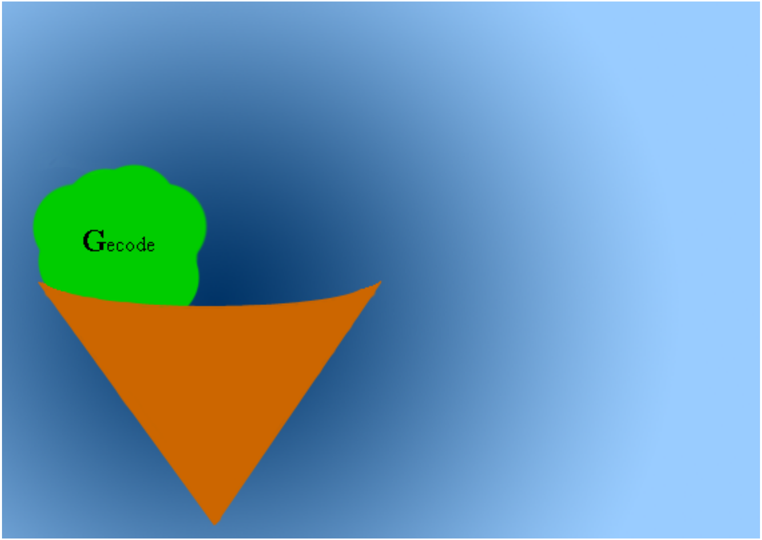
GELATO

IML diagram of the main components



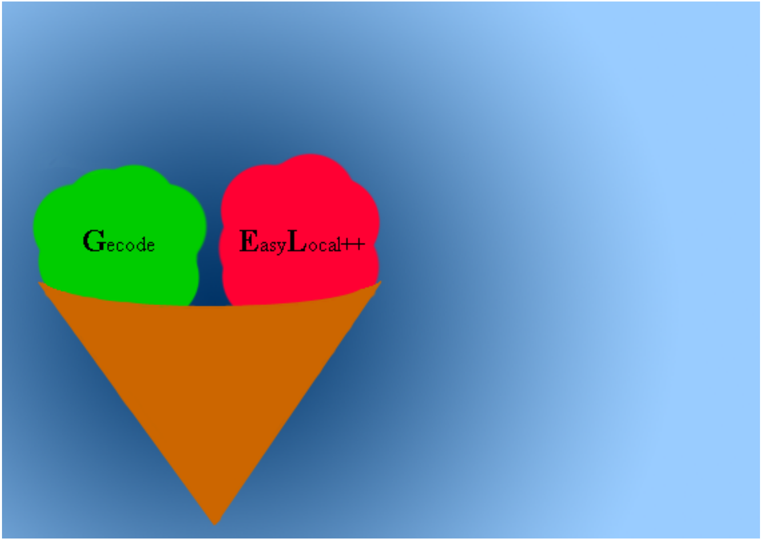
GELATO

IML diagram of the main components



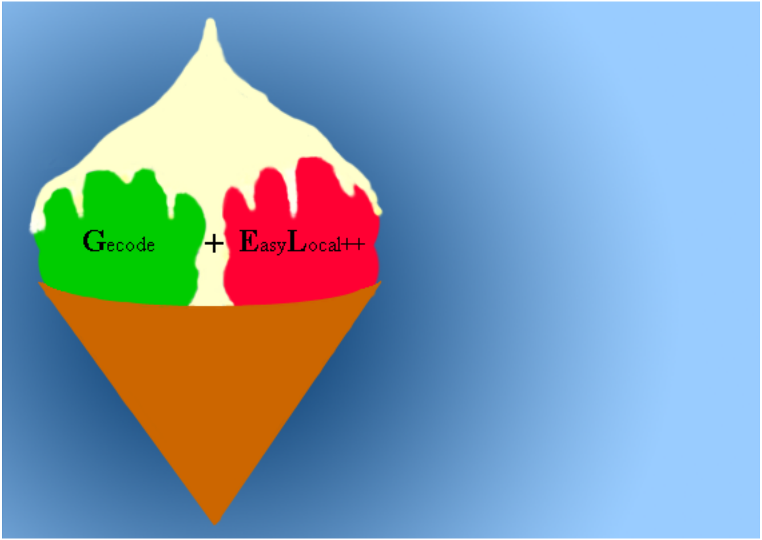
GELATO

IML diagram of the main components



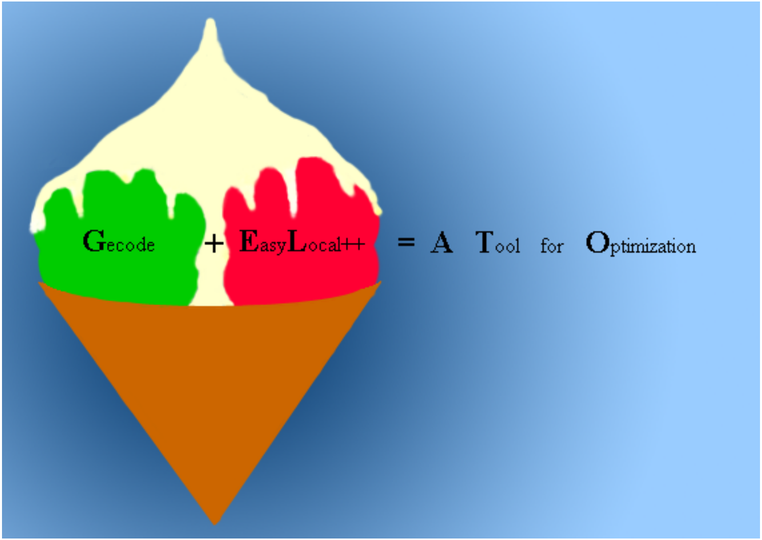
GELATO

IML diagram of the main components



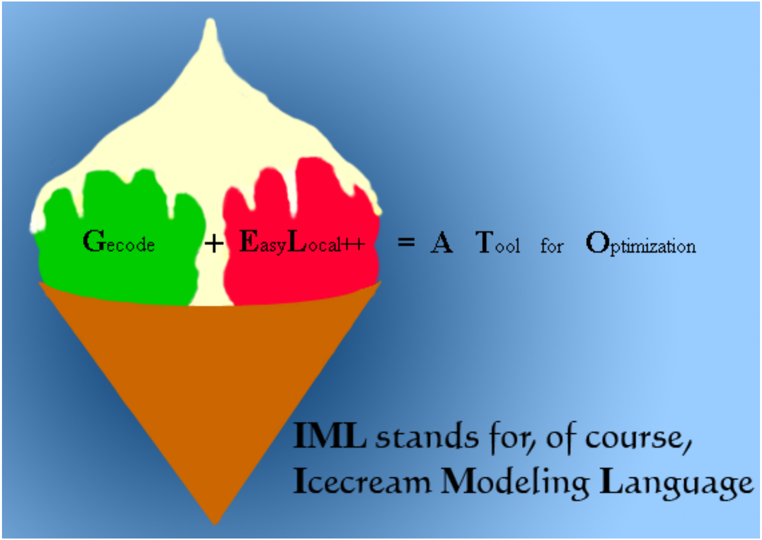
GELATO

IML diagram of the main components



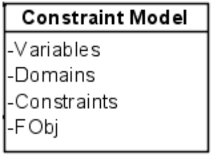
GELATO

IML diagram of the main components



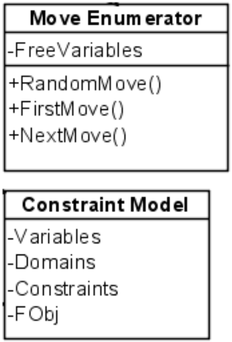
GELATO

UML class diagram of the main components



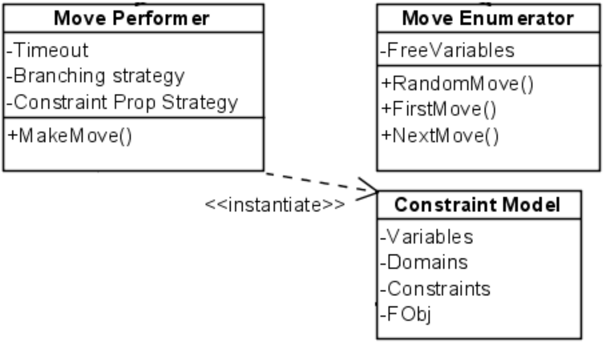
GELATO

UML class diagram of the main components

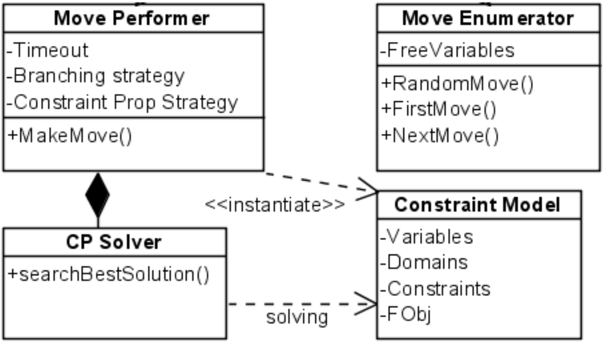


GELATO

UML class diagram of the main components

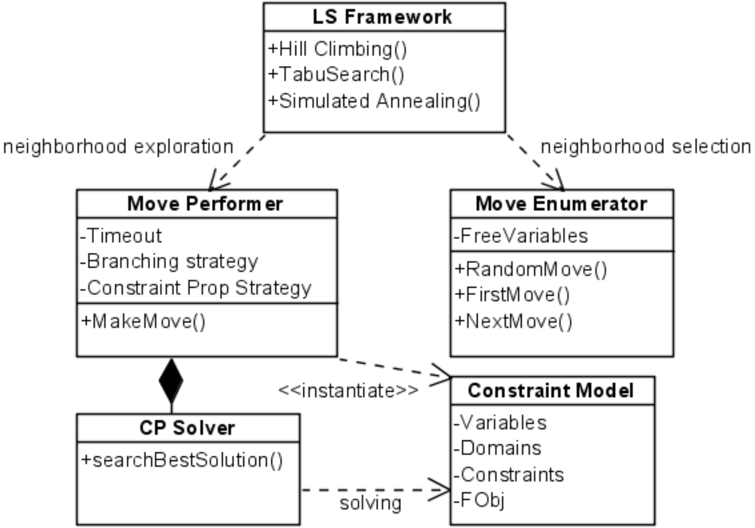


UML class diagram of the main components



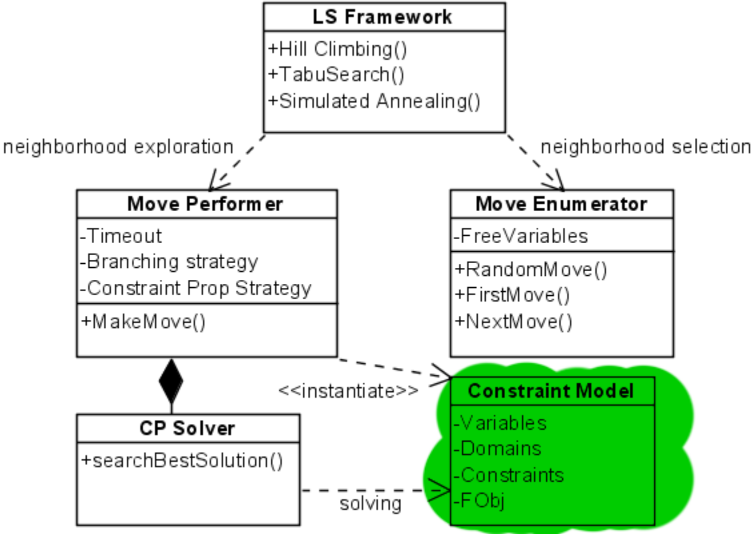
GELATO

UML class diagram of the main components



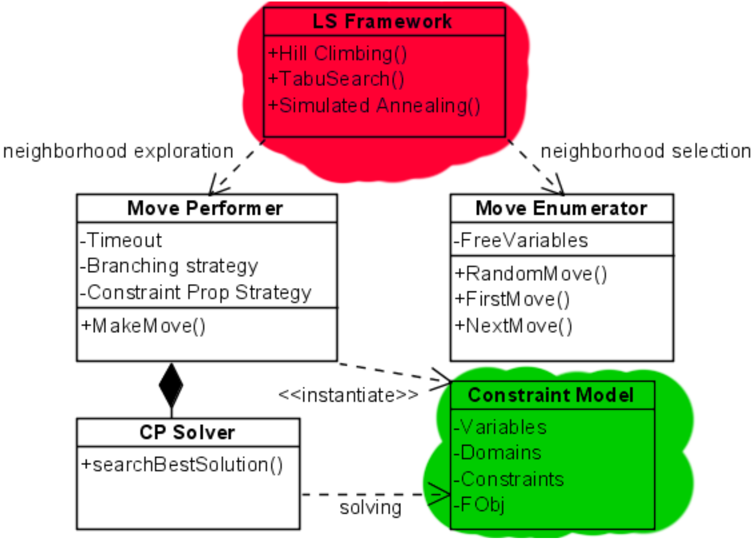
GELATO

UML class diagram of the main components



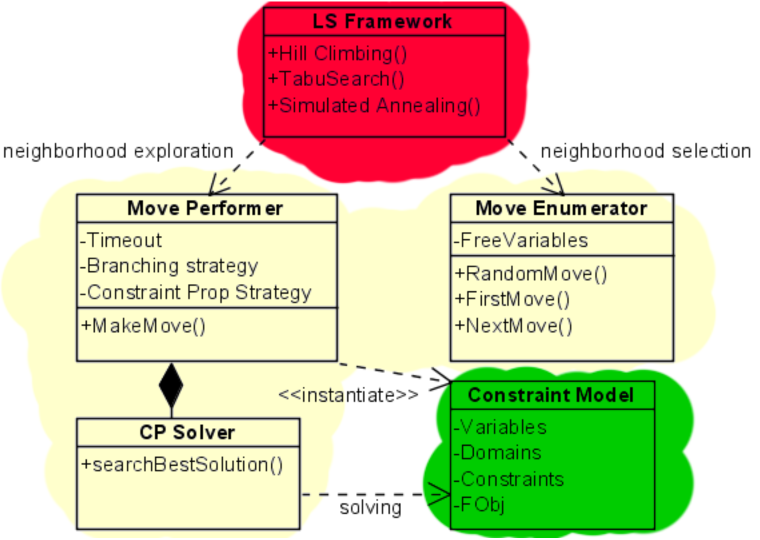
GELATO

UML class diagram of the main components



GELATO

UML class diagram of the main components



Asymmetric Travel Salesman Problem

We tested GELATO on instances of growing sizes of the *ATSP*, taken from the TSPLib

Definition - ATSP

- Given a complete directed graph $G = (V, E)$ and
- a function c that assigns a cost to each edge (i, j),

find a *roundtrip of minimal cost visiting each node once*.
A TSP is *asymmetric* if there exists (i, j) such that $c(i, j) \neq c(j, i)$ (imagine a climbing road).

Let us focus on:

- The Gecode Model
- The Neighborhood Definition and Exploration
- The main Local Search Algorithm

Asymmetric Travel Salesman Problem

We tested GELATO on instances of growing sizes of the *ATSP*, taken from the TSPLib

Definition - ATSP

- Given a complete directed graph $G = (V, E)$ and
- a function c that assigns a cost to each edge (i, j),

find a *roundtrip of minimal cost visiting each node once*.
A TSP is *asymmetric* if there exists (i, j) such that $c(i, j) \neq c(j, i)$ (imagine a climbing road).

Let us focus on:

- The Gecode Model
- The Neighborhood Definition and Exploration
- The main Local Search Algorithm

Asymmetric Travel Salesman Problem

Gecode model

- Model taken from examples of the Gecode package
- We show the possibility of *using* GELATO *starting from existing CP models* (very few changes needed)

Model

- *Variables* X_i, with $i = 0..|V| - 1$
- *Domains* $D_i = 0..(|V| - 1) \forall i$: $X_v = d$ means that vertex v is the d-th visited
- *Constraint*: circuit (every vertices permutation is a feasible solution)
- $fobj = \sum_{(i,j) \in \text{Circuit}} c(i,j)$

Large Neighborhood Definition

- Given a number $N < |V|$ and given a solution
- N variables are randomly selected and left free, while the other ones remain fixed

Large Neighborhood Exploration

- Exhaustive exploration of the N free variables
- Exploration performed via the GECODE model with the following parameters:
  - *variable selection*: smallest domain first
  - *values selection*: random
  - *timeout*: following Comet, we set the maximum number of failures (instead than a time value)
  - *neighborhood returned*: best solution found before timeout

Large Neighborhood Definition

- Given a number $N < |V|$ and given a solution
- N variables are randomly selected and left free, while the other ones remain fixed

Large Neighborhood Exploration

- Exhaustive exploration of the N free variables
- Exploration performed via the GECODE model with the following parameters:
  - *variable selection*: smallest domain first
  - *values selection*: random
  - *timeout*: following Comet, we set the maximum number of failures (instead than a time value)
  - *neighborhood returned*: best solution found before timeout

Asymmetric Travel Salesman Problem

Main Local Search Algorithm

Traditional *Hill Climbing* heuristic, parametric on

- N (neighborhood size)
- K (number of consecutive idle iterations allowed)

Algorithm

- Classical Hill Climbing method
- Large Neighborhoods explored via Gecode , until a timeout expires
- Hill Climbing stopped when stagnation of the algorithm is detected (K idle iteration)

Experiments

Test Benchmarks

- We tested GELATO on TSPlib instances of growing dimension ($|V| = 17, 34, 56, 71, 100, 171$)
- We compare
  - a pure constraint programming approach in Gecode (leftmost, increasing ordering)
  - a pure local search (HC) approach in EasyLocal++
  - different LNS approaches encoded in GELATO
- LNS approaches differ for the neighborhood size (N)
  - 20%, 25%, 30%, 35%, 40%, 45% of $|V|$
- and for the stop criterions for the exploration of a single large neighborhood: maximum number of admitted *failures* equal to
  - $2^{\sqrt{|FV|*0.5}}$
  - $2^{\sqrt{|FV|*1}}$
  - $2^{\sqrt{|FV|*1.5}}$
- number K of max consecutive idle iterations: 50

Experiments

Test Benchmarks

- We tested GELATO on TSPlib instances of growing dimension ($|V| = 17, 34, 56, 71, 100, 171$)
- We compare
  - a pure constraint programming approach in Gecode (leftmost, increasing ordering)
  - a pure local search (HC) approach in EasyLocal++
  - different LNS approaches encoded in GELATO
- LNS approaches differ for the neighborhood size (N)
  - 20%, 25%, 30%, 35%, 40%, 45% of $|V|$
- and for the stop criterions for the exploration of a single large neighborhood: maximum number of admitted *failures* equal to
  - $2^{\sqrt{|FV|*0.5}}$
  - $2^{\sqrt{|FV|*1}}$
  - $2^{\sqrt{|FV|*1.5}}$
- number K of max consecutive idle iterations: 50

Experiments

Test Benchmarks

- We tested GELATO on TSPlib instances of growing dimension ($|V| = 17, 34, 56, 71, 100, 171$)
- We compare
  - a pure constraint programming approach in Gecode (leftmost, increasing ordering)
  - a pure local search (HC) approach in EasyLocal++
  - different LNS approaches encoded in GELATO
- LNS approaches differ for the neighborhood size (N)
  - 20%, 25%, 30%, 35%, 40%, 45% of $|V|$
- and for the stop criterions for the exploration of a single large neighborhood: maximum number of admitted *failures* equal to
  - $2^{\sqrt{|FV| * 0.5}}$
  - $2^{\sqrt{|FV| * 1}}$
  - $2^{\sqrt{|FV| * 1.5}}$
- number K of max consecutive idle iterations: 50

Experiments

Test Benchmarks

- We tested GELATO on TSPlib instances of growing dimension ($|V| = 17, 34, 56, 71, 100, 171$)
- We compare
  - a pure constraint programming approach in Gecode (leftmost, increasing ordering)
  - a pure local search (HC) approach in EasyLocal++
  - different LNS approaches encoded in GELATO
- LNS approaches differ for the neighborhood size (N)
  - 20%, 25%, 30%, 35%, 40%, 45% of $|V|$
- and for the stop criterions for the exploration of a single large neighborhood: maximum number of admitted *failures* equal to
  - $2^{\sqrt{|FV|*0.5}}$
  - $2^{\sqrt{|FV|*1}}$
  - $2^{\sqrt{|FV|*1.5}}$
- number K of max consecutive idle iterations: 50

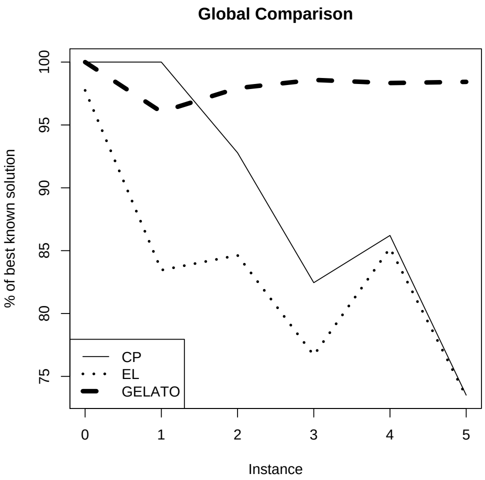
Experiments

Test Benchmarks

- We tested GELATO on TSPlib instances of growing dimension ($|V| = 17, 34, 56, 71, 100, 171$)
- We compare
  - a pure constraint programming approach in Gecode (leftmost, increasing ordering)
  - a pure local search (HC) approach in EasyLocal++
  - different LNS approaches encoded in GELATO
- LNS approaches differ for the neighborhood size (N)
  - 20%, 25%, 30%, 35%, 40%, 45% of $|V|$
- and for the stop criterions for the exploration of a single large neighborhood: maximum number of admitted *failures* equal to
  - $2^{\sqrt{|FV| * 0.5}}$
  - $2^{\sqrt{|FV| * 1}}$
  - $2^{\sqrt{|FV| * 1.5}}$
- number K of max consecutive idle iterations: 50

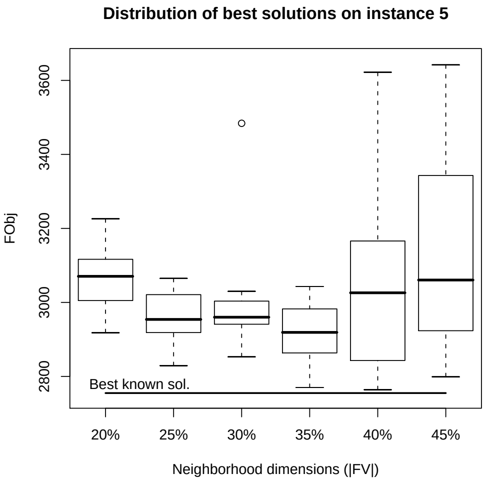
Experiments

Results - Best solutions of CP, EasyLocal++, and GELATO on all instances



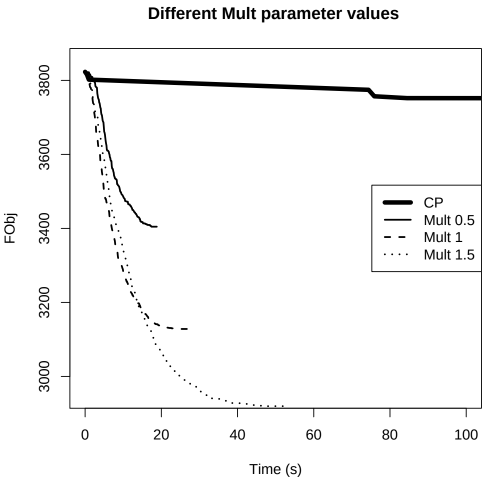
Experiments

Results - Different methods on instance 5 (the hardest one)



Experiments

Results - Behavior of GELATO for different stop criterions



Comparing Gecode with GELATO

- As instances grow, GELATO definitely outperforms Gecode (if we do not necessarily need of finding the optimum solution)
- CP initially finds some improving solutions, but then no other significant improving solutions (exhaustive exploration)
- LNS has a more regular trend: objective function is permanently improved
- The pure LS approach is the weakest one, always outperformed by the others

COMPARING DIFFERENT GELATO SETTINGS

- LNS on small neighborhoods is very fast: very big improvements in some seconds
- LNS on large neighborhoods is slower but it can reach better solutions
- This trade-off must be investigates experimentally
- A stop criterion based on a high limit on number of failures gives better results

Conclusions

Final considerations

- Our GELATO hybrid framework allows to *combine a given CP model into a LS framework* in a straightforward way
- We can *use a* Gecode *model as a base to define (freely) neighborhoods*, using EasyLocal++ to execute *competitive Large Neighborhood Search algorithms*
- We *tested* GELATO *on several instances of the ATSP*: performances of the hybrid LNS algorighms are very faster w.r.t. the pure CP approach
- We are analyzing *a general Multi Neighborhood hybrid meta-heuristic* that should improve the results obtained so far

Conclusions

Final considerations

- Our GELATO hybrid framework allows to *combine a given CP model into a LS framework* in a straightforward way
- We can *use a* Gecode *model as a base to define (freely) neighborhoods*, using EasyLocal++ to execute *competitive Large Neighborhood Search algorithms*
- We *tested* GELATO *on several instances of the ATSP*: performances of the hybrid LNS algorighms are very faster w.r.t. the pure CP approach
- We are analyzing *a general Multi Neighborhood hybrid meta-heuristic* that should improve the results obtained so far

Future works

Before CPAIOR deadline

1. Developing and testing new hybrid algorithms, on new problems (wip on: Sudoku, MEB, PF), and other LS strategies
2. Deep comparison wrt Comet (on same models, same parameters)
3. Embedding GELATO into a general modeling framework. We already developed (ICLP2008) a front-end from SICStus Prolog and Minizinc to Gecode.

Future works

Before CPAIOR deadline

1. Developing and testing new hybrid algorithms, on new problems (wip on: Sudoku, MEB, PF), and other LS strategies
2. Deep comparison wrt Comet (on same models, same parameters)
3. Embedding GELATO into a general modeling framework. We already developed (ICLP2008) a front-end from SICStus Prolog and Minizinc to Gecode.

Future works

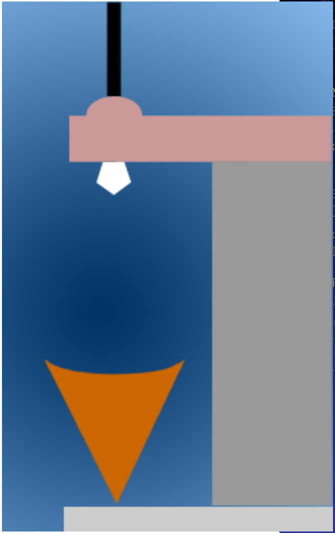
Before CPAIOR deadline

1. Developing and testing new hybrid algorithms, on new problems (wip on: Sudoku, MEB, PF), and other LS strategies
2. Deep comparison wrt Comet (on same models, same parameters)
3. Embedding GELATO into a general modeling framework. We already developed (ICLP2008) a front-end from SICStus Prolog and Minizinc to Gecode.

Future works

Before CPAIOR deadline

1. Developing and testing new hybrid algorithms, on new problems (wip on: Sudoku, MEB, PF), and other LS strategies
2. Deep comparison wrt Comet (on same models, same parameters)
3. Embedding GELATO into a general modeling framework. We already developed (ICLP2008) a front-end from SICStus Prolog and Minizinc to Gecode.



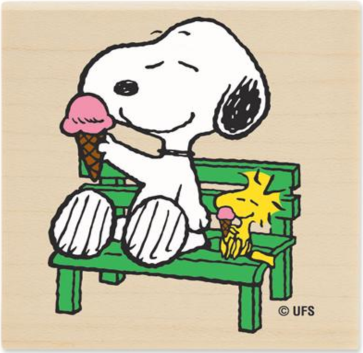
Future works

Before CPAIOR deadline

1. Developing and testing new hybrid algorithms, on new problems (wip on: Sudoku, MEB, PF), and other LS strategies
2. Deep comparison wrt Comet (on same models, same parameters)
3. Embedding GELATO into a general modeling framework. We already developed (ICLP2008) a front-end from SICStus Prolog and Minizinc to Gecode.

That's all!

Thank you for your attention



Drafts

Experiments

A possible meta-heuristic

A possible meta-heuristic came out from these considerations:

Algorithm

1. start with small LNS (in this way we try to get the best improvement in the shortest time);
2. when no better solutions can be found, increase the neighborhood's size, then launch Large Neighborhood Search;
3. iterate 2 since the neighborhood's sizes increase to intractable/ineffective ones.

Hybridization of CP and LS

Classical Master-Slave Approaches

- Master algorithm based on CP improved by LS at some point, e.g.:
    1. at a *leaf* or *internal node* of the search tree, to improve the solution found
    2. at a *node* of the search tree, to restrict the list of child-nodes
    3. to generate in a greedy way a *path* in the search tree
- Master LS algorithm can benefit of the support of CP, e.g.:
    1. to *analyze the neighborhood* and discarding the neighbors not satisfying the constraints
    2. to *explore a fragment of the neighborhood* of the current sol
    3. to *define the search of the best neighbor* as a problem of constrained optimization (COP)

- Master algorithm based on CP improved by LS at some point, e.g.:
  1. at a *leaf* or *internal node* of the search tree, to improve the solution found
  2. at a *node* of the search tree, to restrict the list of child-nodes
  3. to generate in a greedy way a *path* in the search tree
- Master LS algorithm can benefit of the support of CP, e.g.:
  1. to *analyze the neighborhood* and discarding the neighbors not satisfying the constraints
  2. to *explore a fragment of the neighborhood* of the current sol
  3. to *define the search of the best neighbor* as a problem of constrained optimization (COP)

Hybridization of CP and LS

Free Approaches

Hybrid paradigms not based on the Master-Slave philosophy have been proposed:

- In [Monfroy-Frederic-Lambert ICLP 2004] LS and CP are broken up into their component parts managed at the same level:
  - neighborhood exploration
  - constraint propagation
  - variable assignment
- In [Van Hentenryck-Michel 2005] CP and LS are combined in a programming language (COMET), that supports both modeling and search abstractions, and where *CP is used to describe and control local search*