GHOST: A stealth solver

Florian Richoux - Jean-François Baffier - Alberto Uriarte

October 4th, 2014

UNIVERSITÉ DE NANTES

東京大学
THE UNIVERSITY OF TOKYO

Drexel
UNIVERSITY

Outline

Outline

CSP/ COP

Constraint Satisfaction Problems (CSP)

CSP is a homogeneous framework to model combinatorial problems.

Constraint Optimization Problems (COP)

Same for optimization problems.

Examples

- AI,
- Graph problems,
- Database problems,
- Bio-informatics,
- ...

Definition of a CSP

A CSP is defined by a tuple (V, D, C) such that:

$$\text{CSP} = \left[\begin{array}{ll} V: & \textit{Set of variables.} \\ D: & \textit{Domain (set of values of variables).} \\ C: & \textit{Set of constraints.} \end{array} \right.$$

Definition of a CSP

A CSP is defined by a tuple (V, D, C) such that:

$$\text{CSP} = \left[\begin{array}{ll} V: & \textit{Set of variables.} \\ D: & \textit{Domain (set of values of variables).} \\ C: & \textit{Set of constraints.} \end{array} \right.$$

CSP

- V = {x, y, z}
- D = {0, 1}
- C = {=, $\neq$}

Problem

Definition of a CSP

A CSP is defined by a tuple (V, D, C) such that:

$$\text{CSP} = \left[\begin{array}{ll} V: & \textit{Set of variables.} \\ D: & \textit{Domain (set of values of variables).} \\ C: & \textit{Set of constraints.} \end{array}\right.$$

CSP

- $V = \{x, y, z\}$
- $D = \{0, 1\}$
- $C = \{=, \neq\}$

⇐ Problem

CSP formula (aka CSP instance)

$$(z = y) \wedge (y \neq x) \wedge (x = z)$$

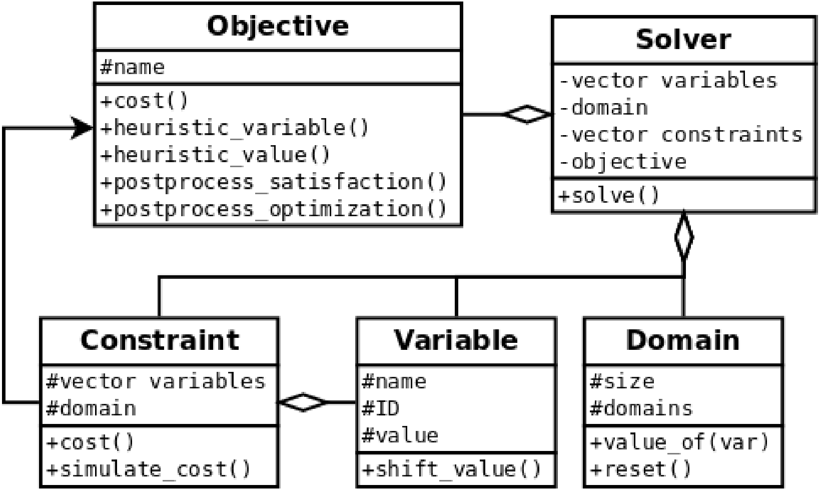
⇐ Problem instance

Outline

GHOST is a templated C++ solver aiming two users:

- The **casual user**, using GHOST to solve an already encoded problem.
- The **developer user**, who wants to use GHOST to implement and solve his/her problem.

Classes of GHOST



Casual user

```
// Variables
vector< Unit > variables = { ... }

// Specific to the target selection problem
vector< UnitEnemy  > enemies  = {  ... }

// Domain
TargetDomain domain( variables.size(), &enemies );

// Constraints
vector< shared_ptr<TargetConstraint> > constraints
{ make_shared<TargetConstraint>( &variables, &domain ) };

// Objective
shared_ptr<TargetObjective> objective = make_shared<MaxDamage>();

// Solver
Solver<Unit, TargetDomain, TargetConstraint>
  solver( &variables, &domain, constraints, objective );

// Call Solver::solve
solver.solve();
```

Uses of GHOST

Developer user

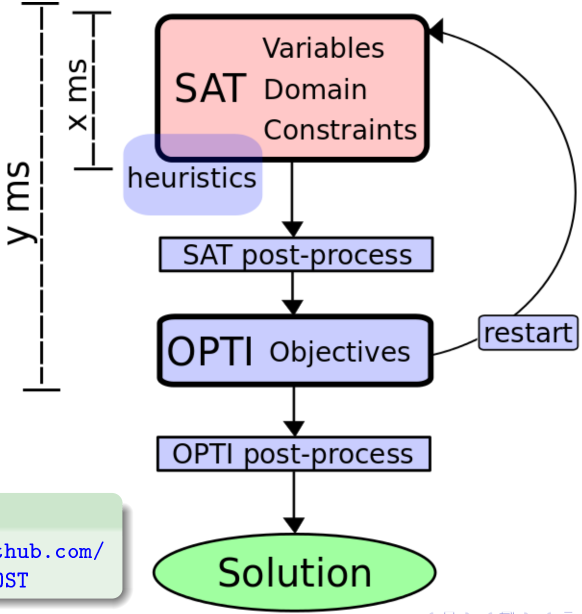
```
// Define its own variables
class Unit : public Variable {...}

// Define its own domain
class TargetDomain : public Domain<Unit>  {...}

// Define its own constraints
class TargetConstraint : public Constraints<Unit, TargetDomain>  {...}

// Define its own objectives
class TargetObjective : public Objective<Unit, TargetDomain>  {...}
```

Architecture of GHOST



Source code

https://github.com/richoux/GHOST

Outline

RTS problems

Problem families

Strategy

Tactic

Reactive Control

RTS problems

Problem families

Strategy

Tactic

Reactive Control

Example of problems

Build order planning

Wall-in

Target selection

Reactive control: Target selection problem

Target selection problem: Statement

- Range,
- Cooldown,
- Damage efficiency.

Target selection problem: Statement

- Range,
- Cooldown,
- Damage efficiency.

Damage efficiency

		Damage type		
		Concussive	Normal	Explosive
size	Small	100%	100%	50%
	Medium	50%	100%	75%
	Large	25%	100%	100%

CSP/ COP model

- V = (Sub)set of your units.
- D = A group of enemy units.
- C = Each living, ready-to-shoot unit must aim a living enemy unit within its range, if any.

CSP/ COP model

- V = (Sub)set of your units.
- D = A group of enemy units.
- C = Each living, ready-to-shoot unit must aim a living enemy unit within its range, if any.

COP objectives

- **Max damage**, where our group tries to deal as much damage as possible within the current frame.
- **Max kill**, where our group tries to kill as much enemy units as possible within the current frame.

Experiments

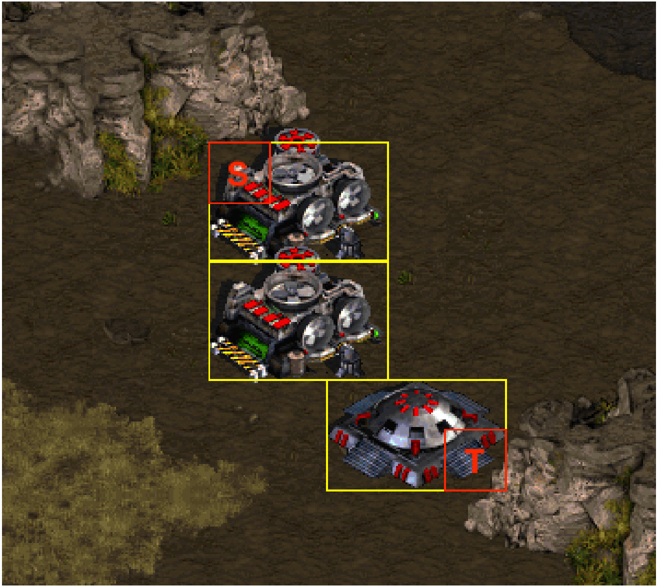
Mean of 100 runs for each objectives. SAT-Opti Timeouts: 1-3ms and 2-5ms.

				GHOST victory	
	Objective	Win %	# Draws	# GHOST living units	GHOST HP
3ms	Max Damage	86.0	3	2.8	153.4
3ms	Max Kill	80.0	2	3.0	163.2
5ms	Max Damage	91.0	1	2.8	149.6
5ms	Max Kill	82.0	3	3.0	169.5

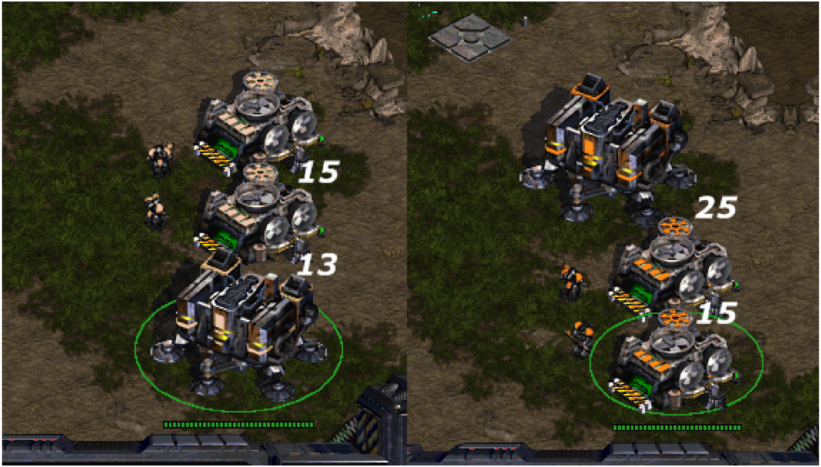
		Opponent victory	
	Objective	# Opponent living units	Opponent HP
3ms	Max Damage	1.0	37.1
3ms	Max Kill	1.1	62.8
5ms	Max Damage	1.1	58.7
5ms	Max Kill	1.0	37.2

Tactic:
Wall-in problem

Wall-in problem: Statement



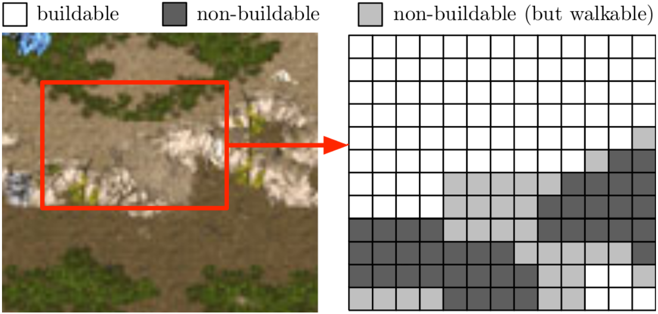
Wall-in problem: Statement



Wall-in problem: Model

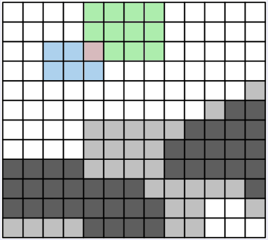
CSP/ COP model

- V = Set of your buildings.
- D = Positions around a choke (in a 16×12 rectangle).



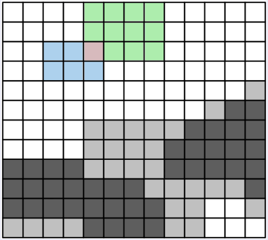
Wall-in problem: Model

Overlap

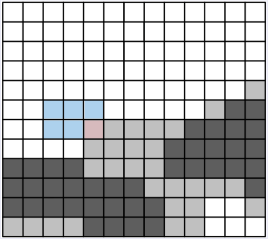


Wall-in problem: Model

Overlap

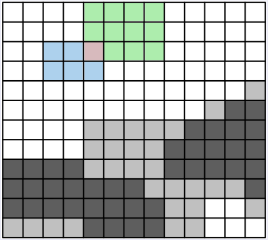


Buildable

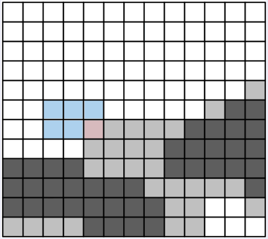


Wall-in problem: Model

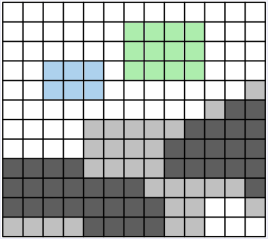
Overlap



Buildable

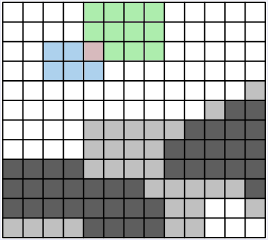


NoHoles

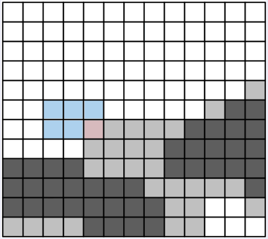


Wall-in problem: Model

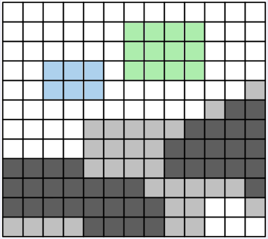
Overlap



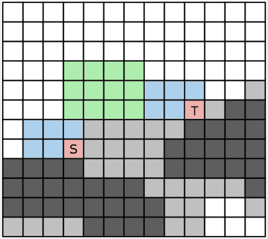
Buildable



NoHoles



StartingTargetTile



Wall-in problem: Model

CSP Wall-in question

What is the position of my buildings to make a wall?
(and what are the buildings I use?)

Wall-in problem: Model

CSP Wall-in question

What is the position of my buildings to make a wall?
(and what are the buildings I use?)

COP Wall-in question

Optimize an objective function for my wall:

- Make a wall using the smallest number of buildings.
- Make a wall using the lowest technology.
- Make a wall with the fewest number of gaps.

Wall-in problem: Results

100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).

Wall-in problem: Results

100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).

CSP results: making a wall with 20ms runs

#Runs	1	2	3	4
Average Cost	1.59	0.58	0.33	0.18
Solved %	45.83%	71.10%	80.70%	88.45%

#Runs	5	10	20	50
Average Cost	0.13	0.03	0.01	0.0006
Solved %	91.58%	97.23%	99.18%	99.95%

Wall-in problem: Results

100 tests over 48 chokes from 7 StarCraft classic maps (so 4800 tests).

CSP results: making a wall with 20ms runs

#Runs	1	2	3	4
Average Cost	1.59	0.58	0.33	0.18
Solved %	45.83%	71.10%	80.70%	88.45%

#Runs	5	10	20	50
Average Cost	0.13	0.03	0.01	0.0006
Solved %	91.58%	97.23%	99.18%	99.95%

COP results: optimize a wall within 150ms (with a SAT timeout of 20ms)

	Satisfaction run	Optimization run	Optimization run solved
#Buildings	3.12	2.65	96.83%
#Gaps	1.19	0.05	96.79%
Low tech	1.95	1.56	95.87%

Wall-in problem: Results

More results for #Gaps

Over 4800 tests:

- the solver found 4646 walls (96.8%) within 150ms,
- whose 4400 were perfect, i.e., gap-free.(94.7% of walls).

Strategy: Build order planning problem

Build order problem: Statement

BO: A series of actions following a specific timing, in order to achieve a goal.

Dependencies

Actions may have dependencies.

Example

To build a Factory, you need first a Barracks.

Build order problem: Model

BO can be modeled as a **permutation** problem: Huge search space reduction.

Build order problem: Model

BO can be modeled as a **permutation** problem: Huge search space reduction.

CSP/ COP model

- V = All actions you need to reach your goal.
- D = Order of actions.
- C = Each dependency of an action α must occurs before α.

Build order problem: Model

BO can be modeled as a **permutation** problem: Huge search space reduction.

CSP/ COP model

- V = All actions you need to reach your goal.
- D = Order of actions.
- C = Each dependency of an action α must occurs before α.

COP objective

Minimize the makespan.

Build order problem: Simulator

StarCraft simulator inside GHOST

We code a SC simulator to emulate resources gathering, units producing (including workers), supply capacity, constructions, etc.

StarCraft simulator inside GHOST

We code a SC simulator to emulate resources gathering, units producing (including workers), supply capacity, constructions, etc.

- Time to go build something: **5t**
- Time to go back gathering minerals after building something: **4t**
- Time to go from the base to mineral patches to start mining: **5t**
- Time for a worker to switch from mineral to gas: **5t**
- Mineral gathering rate: **0.68 mineral** per worker per t
- Gas gathering rate: **1.15 gas** per worker per t

Build order problem: Results

Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

Mean of 10 runs for each BO.

SAT-Opti Timeouts: 20-30ms.

Build order problem: Results

Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

Mean of 10 runs for each BO.

SAT-Opti Timeouts: 20-30ms.

Games till 10.000 frames				
Match-up	Humans	GHOST	% solved	Gain
All	656.02	619.62	94.4	**36.40**
PvP	651.51	608.06	95.0	**43.45**
PvT	660.50	628.17	93.9	**32.33**
PvZ	649.20	609.34	95.0	**39.86**
All pro	643.38	597.25	96.3	**46.13**

Build order problem: Results

Two test sets

1. 3,647 BOs from TeamLiquid, GosuGamers and ICCup.
2. 8 BOs from top Korean pro-gamers.

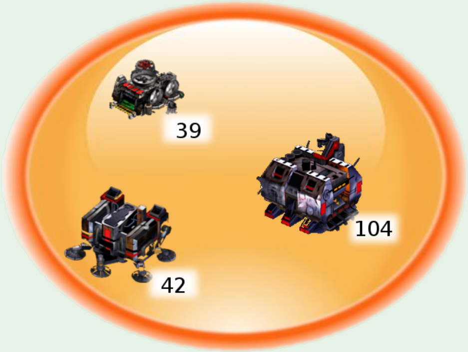
Mean of 10 runs for each BO.

SAT-Opti Timeouts: 20-30ms.

Games till 7.800 frames				
Match-up	Humans	GHOST	% solved	Gain
All	522.50	491.11	98.8	**31.39**
PvP	516.08	485.56	99.3	**30.52**
PvT	527.66	506.65	98.3	**21.01**
PvZ	515.81	458.23	99.7	**57.58**
All pro	506.38	480.88	100	**25.50**

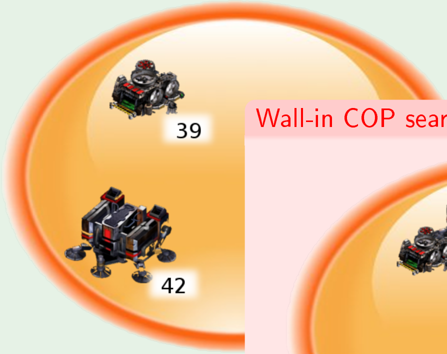
Why searching for walls takes more time than build orders?

Wall-in CSP search space



Why searching for walls takes more time than build orders?

Wall-in CSP search space



Wall-in COP search space

Outline

Conclusion

GHOST is:

- **General**: Can deal with any CSP/COP models without parameter tuning or code optimization needed.
- **Easy**: Both for casual and developer users.
- **Efficient**: We obtained good results on different RTS problems.
- **Open**: Contributions welcome!

Discussion

Future works

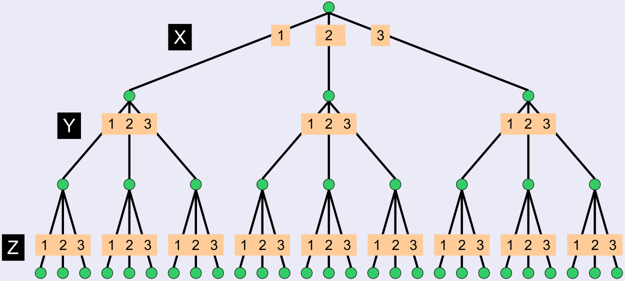
- GHOST will be a library using BWAPI 4.
- Implement a pause/resume system for long computations.
- Parallelize the solver exploiting the available cores.
- CSP/COP cannot deal with uncertainty. Think to a new formalism.

Questions?

florian.richoux@univ-nantes.fr

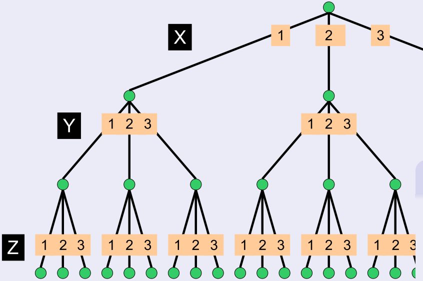
@FloRicx

Complete solvers

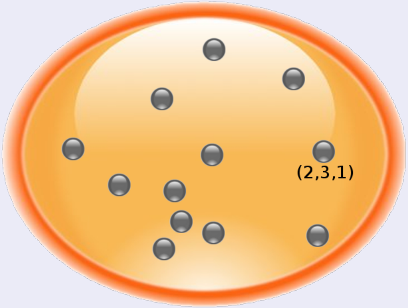


Complete Vs. Incomplete solvers

Complete solvers

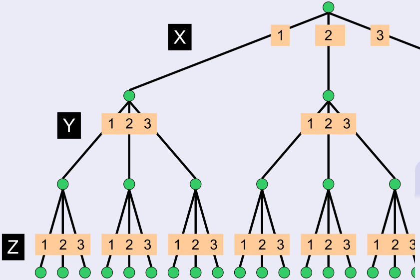


Meta-heuristics

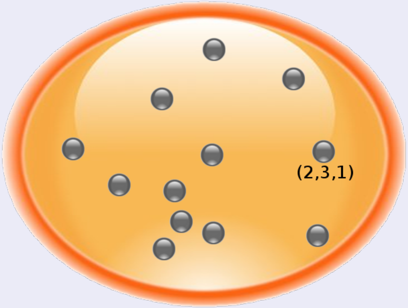


Complete Vs. Incomplete solvers

Complete solvers

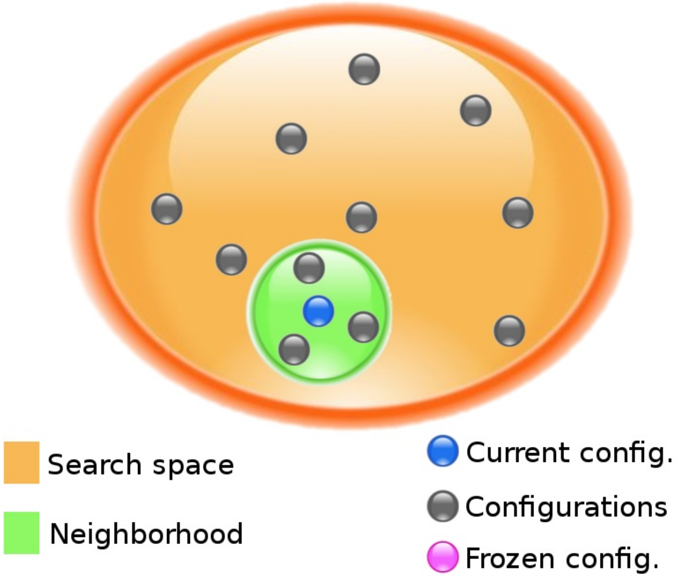


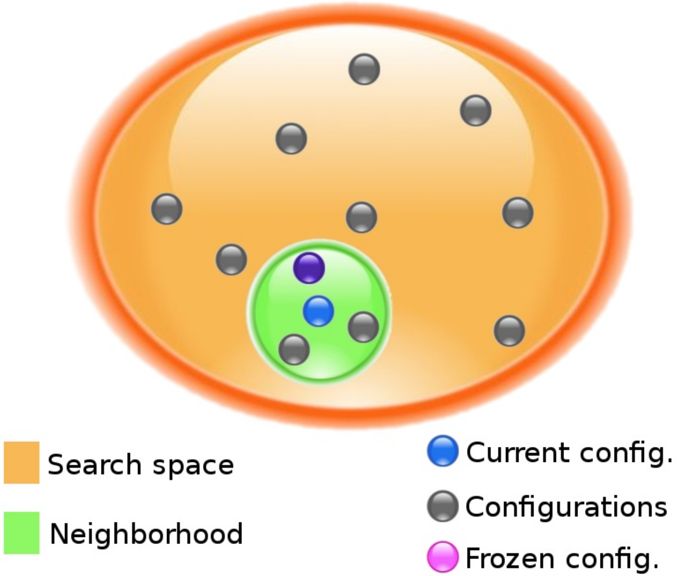
Meta-heuristics

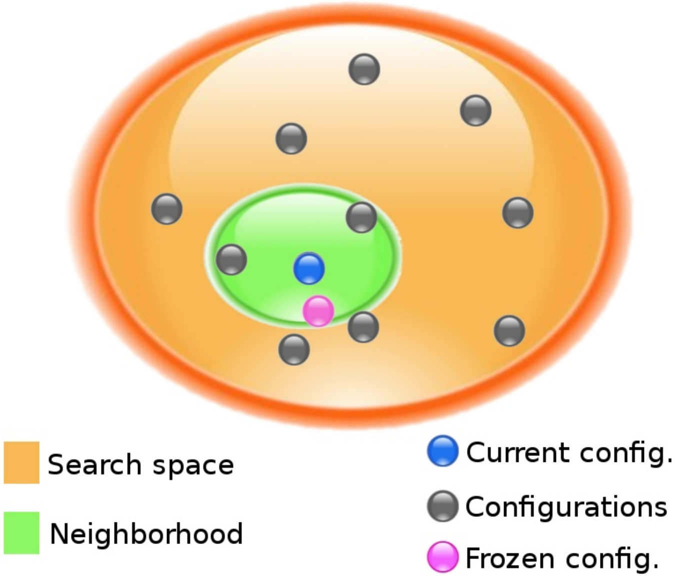


- Way faster
- Can deal with big size problems

- Not sure to find a solution
- Lost if no solutions


Search space
Neighborhood
Current config.
Configurations
Frozen config.


Search space
Neighborhood
Current config.
Configurations
Frozen config.


Search space
Neighborhood
Current config.
Configurations
Frozen config.

Main idea

Constraint $x = y$ $\Rightarrow$ $\underbrace{(x = y)}_{C_1} \wedge \underbrace{(y = z)}_{C_2}$

Adaptive Search [Codognet Diaz 2001]

Main idea

Constraint $x = y$

$\Downarrow$

Error function which measures how much the constraint is satisfied
$\mid x - y \mid$

$\Rightarrow$ $\underbrace{(x = y)}_{C_1} \wedge \underbrace{(y = z)}_{C_2}$

$\Rightarrow$ Assume we have $x = 3,\ y = 6,\ z = 1$

Error(C_1) $= 3$
Error(C_2) $= 5$

Adaptive Search [Codognet Diaz 2001]

Main idea

Constraint $x = y$ $\Rightarrow$ $\underbrace{(x = y)}_{C_1} \wedge \underbrace{(y = z)}_{C_2}$

$\Downarrow$

Error function which measures how much the constraint is satisfied
$| \, x - y \, |$

$\Rightarrow$ $\left\{ \begin{array}{l} \text{Assume we have} \\ x = 3, \;\; y = 6, \;\; z = 1 \\ \\ \text{Error}(C_1) = 3 \\ \text{Error}(C_2) = 5 \end{array} \right.$

$\Downarrow$

Projection (usually addition of errors) on each variables

$\Rightarrow$ $\left\{ \begin{array}{ll} \text{Error}(x) & = 3 \\ \text{Error}(y) & = 3 + 5 = 8 \\ \text{Error}(z) & = 5 \end{array} \right.$