Solving Combinatorial Problems on HPC with BOBPP

Tarek Menouer, Bertrand Le Cun and Pascal Vander-Swalmen

University of Versailles Saint-Quentin-en-Yvelines, France
PRiSM laboratory

November 16, 2012
WOLFHPC 2012

SC12
Salt Lake City, Utah

Outline

Outline

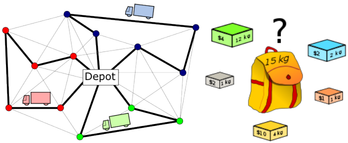
Context of our work

Combinatorial Optimization

- Find an assignment to some variables such that
  - The value of a certain function is minimized (or maximized)
  - Satisfying some constraints

Academic examples

- Vehicle Routing Problem
- Knapsack Problem
- Travelling Salesman Problem



Combinatorial Optimization Problems

Combinatorial Explosion: The Travelling Salesman Problem

- Brut Force: TSP, 20 cities, 20! tours, 10^{-9} s/tour = 77.14 years
- Parallelization with 1,000 cores and 50 % efficiency same problem: 10^{-9} seconds/tour = 56,31 days

Combinatorial Explosion

- Operational Research (OR)
  Use OR methods to reduce the size of search space
  Evaluation/Prunning of partial solutions
- Parallelization
  Parallelization to solve quicker the problem with greater size

Exact resolution up to 15112 cities

- Specific Parallel Branch & Cut: Concorde
- 22,6 years equivalent sequential time by Applegate, Bixby, Chvátal and Cook

Combinatorial Optimization Problems

Combinatorial Explosion: The Travelling Salesman Problem

- Brut Force: TSP, 20 cities, 20! tours, 10^{-9} s/tour $=$ 77.14 years
- Parallelization with 1,000 cores and 50 % efficiency same problem: 10^{-9} seconds/tour $=$ 56,31 days

Combinatorial Explosion

- Operational Research (OR)

  Use OR methods to reduce the size of search space

  Evaluation/Prunning of partial solutions

- Parallelization

  Parallelization to solve quicker the problem with greater size

Exact resolution up to 15112 cities

- Specific Parallel Branch & Cut: Concorde
- 22,6 years equivalent sequential time by Applegate, Bixby, Chvátal and Cook

Combinatorial Optimization Problems

Combinatorial Explosion: The Travelling Salesman Problem

- Brut Force: TSP, 20 cities, 20! tours, 10^{-9} s/tour $=$ 77.14 years
- Parallelization with 1,000 cores and 50 % efficiency same problem: 10^{-9} seconds/tour $=$ 56,31 days

Combinatorial Explosion

- Operational Research (OR)

  Use OR methods to reduce the size of search space

  Evaluation/Prunning of partial solutions

- Parallelization

  Parallelization to solve quicker the problem with greater size

Exact resolution up to 15112 cities

- Specific Parallel Branch & Cut: Concorde
- 22,6 years equivalent sequential time

by Applegate, Bixby, Chvátal and Cook

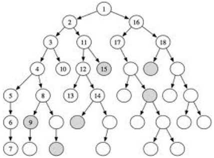
Methods

Exact Algorithms

- Implicit total enumeration
- Tree search-space
- Divide&Conquer, Branch&Bound, Branch&Cut, ...
- Constraint programming

Heuristic Algorithms

- Not exhaustive
- Greedy heuristics
- Local search
- Meta : Simulated Annealing, Tabu search, Genetic algorithms, ...



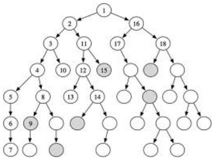
Methods

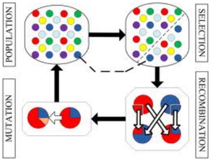
Exact Algorithms

- Implicit total enumeration
- Tree search-space
- Divide&Conquer, Branch&Bound, Branch&Cut, ...
- Constraint programming

Heuristic Algorithms

- Not exhaustive
- Greedy heuristics
- Local search
- Meta : Simulated Annealing, Tabu search, Genetic algorithms, ...

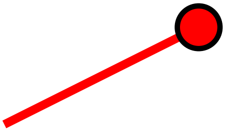




Usefull functionalities for exact methods

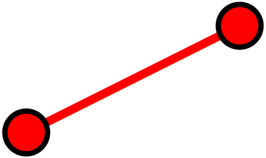
The user must **define** the problem, mainly the data stored in the node.

Usefull functionalities for exact methods



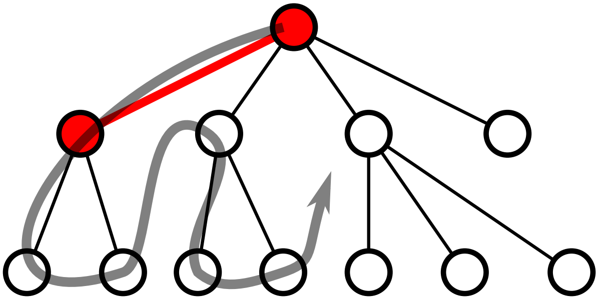
The user **describes** how a node is generated from a parent node (the child generation) according to the branching strategy

Usefull functionalities for exact methods



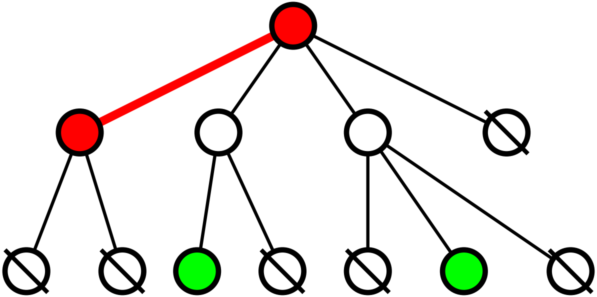
The user **describes** a work on node: evaluation function, constraint propagation, . . .

Usefull functionalities for exact methods



The user may **choose** between different exploration strategies (best first, depth-first search, etc)

Usefull functionalities for exact methods



The user may choose what is the goal of the search : the best solution, the number of feasible/best solutions. that implicates the stop criteria.

Outline

Related Works

Frameworks Solving Exact Combinatorial Optimization problems

- Space search based algorithms
- Parallel Programming environments

Searches

- Divide & Conquer
- Branch & X
- A*
- ...

Programming Environments

- PThreads
- MPI
- KAAPI
- ...

Usual Frameworks

Designed for one algorithm and for one specific programming environment

Related Works

Frameworks Solving Exact Combinatorial Optimization problems

- Space search based algorithms
- Parallel Programming environments

Searches

- Divide & Conquer
- Branch & X
- A*
- ...

Programming Environments

- PThreads
- MPI
- KAAPI
- ...

Usual Frameworks

Designed for one algorithm and for one specific programming environment

Frameworks issues

Several parallelizations

- Two algorithms may have two different behaviours
- There does not exist only one ultimate parallelization

Several parallel environments

Different parallel architectures $\Rightarrow$ different parallel algorithms and libraries

- Shared memory: threads and mutex
- Distributed memory : processes and messages
- Cluster of SMPs: threads and processes
- ...

BOBPP

Objectives

- Solve Combinatorial Optimization Problems based on search-trees
  - Divide & Conquer
  - Branch & Bound
  - Branch & Cut
  - Branch & Price
- Framework which proposes an interface to write such algorithms in sequential and parallel

Functionalities for solving combinatorial problems

- Search strategies: depth-first, breadth-first, best-first, "best of the deepest"-first, . . .
- Aims: best solution, number of best solutions, . . .

BOBPP

Objectives

- Solve Combinatorial Optimization Problems based on search-trees
  - Divide & Conquer
  - Branch & Bound
  - Branch & Cut
  - Branch & Price
- Framework which proposes an interface to write such algorithms in sequential and parallel

Functionalities for solving combinatorial problems

- Search strategies: depth-first, breadth-first, best-first, "best of the deepest"-first, . . .
- Aims: best solution, number of best solutions, . . .

BOBPP

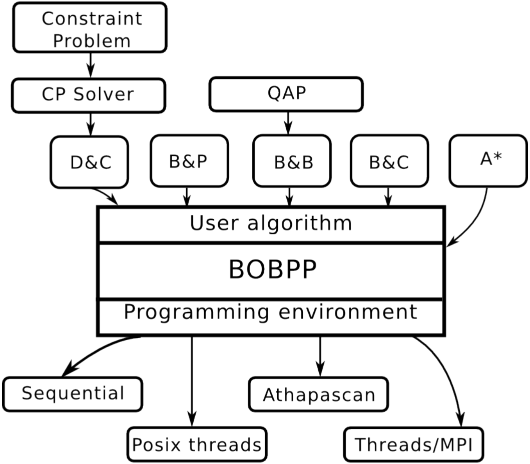
Objectives

- Solve Combinatorial Optimization Problems based on search-trees
    - Divide & Conquer
    - Branch & Bound
    - Branch & Cut
    - Branch & Price
- Framework which proposes an interface to write such algorithms in sequential and parallel

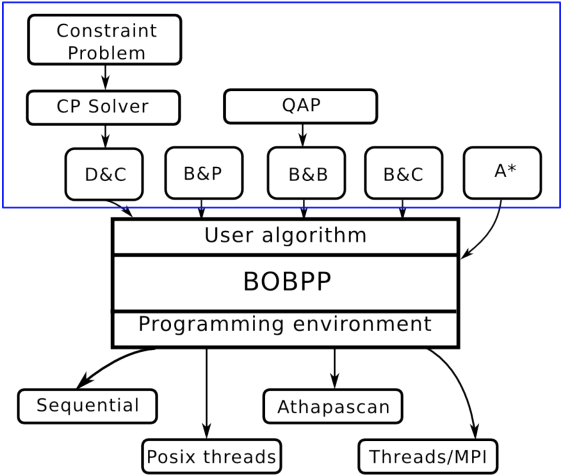
Functionalities for solving combinatorial problems

- Search strategies: depth-first, breadth-first, best-first, "best of the deepest"-first, . . .
- Aims: best solution, number of best solutions, . . .

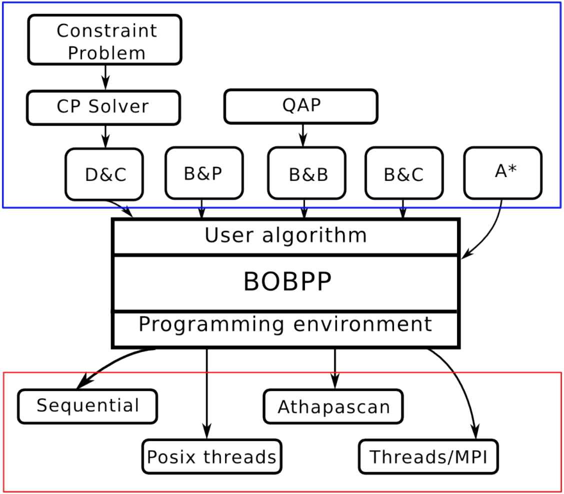
BOBPP


Constraint Problem
CP Solver
QAP
D&C
B&P
B&B
B&C
A*
User algorithm
BOBPP
Programming environment
Sequential
Athapascan
Posix threads
Threads/MPI

BOBPP


Constraint Problem
CP Solver
QAP
D&C
B&P
B&B
B&C
A*
User algorithm
BOBPP
Programming environment
Sequential
Posix threads
Athapascan
Threads/MPI

BOBPP


Constraint Problem
CP Solver
QAP
D&C
B&P
B&B
B&C
A*
User algorithm
BOBPP
Programming environment
Sequential
Athapascan
Posix threads
Threads/MPI

Frameworks comparison

	Algorithms					Environments		
Framework	B&B	B&B+LP	B&P	B&C	D&C	THR	MPI	Hybrid
PPBB	X						X	
BCP		X	X	X			X	
Symphony		X	X	X		X	X	
PEBBL	X						X	
CBC		X	X	X			X	
BOBPP	X	X	X	X	X	X	X	X

BOBPP sources

`http://forge.prism.uvsq.fr/projects/bobpp`

BOBPP, Objects

Classes Involved

Common functionalities:

Class Node:	*	Represents a node of the search-tree
Class Genchild:	*	Methods generating the children of the nodes
Class Instance:	*	Stores all the global data used during the search
Class Goal:		Stores the solution
Class Algo:		Method for the main loop
Class Priority Queue:		Stores all the nodes during the search

Redefined by the user for his specific problem

Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**

BOBPP, Objects

Classes Involved

Common functionalities:

Class Node:	*	Represents a node of the search-tree
Class Genchild:	*	Methods generating the children of the nodes
Class Instance:	*	Stores all the global data used during the search
Class Goal:		Stores the solution
Class Algo:		Method for the main loop
Class Priority Queue:		Stores all the nodes during the search

Redefined by the user for his specific problem

Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**

BOBPP, Objects

Classes Involved

Common functionalities:

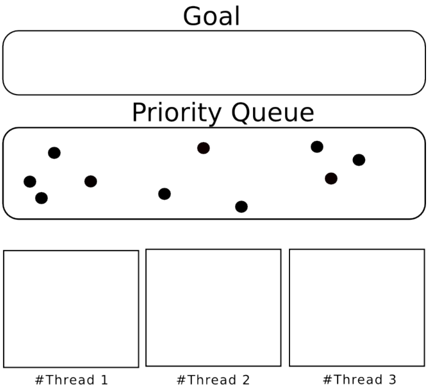
Class Node:	*	Represents a node of the search-tree
Class Genchild:	*	Methods generating the children of the nodes
Class Instance:	*	Stores all the global data used during the search
Class Goal:		Stores the solution
Class Algo:		Method for the main loop
Class Priority Queue:		Stores all the nodes during the search

Redefined by the user for his specific problem

Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**

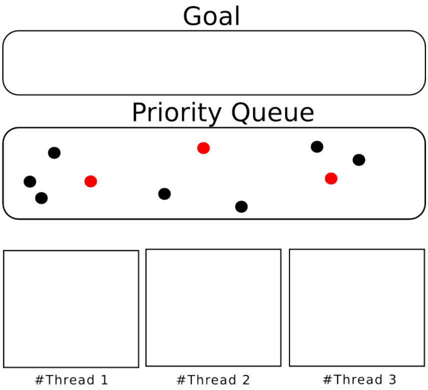
BOBPP, Priority Queue Principle



Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue

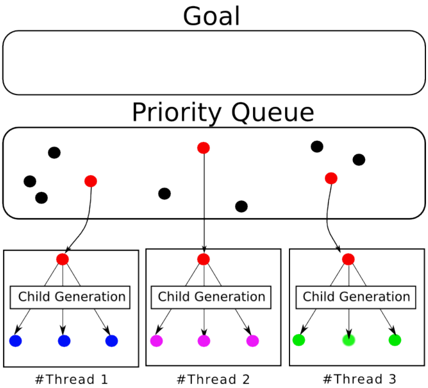
BOBPP, Priority Queue Principle



Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue

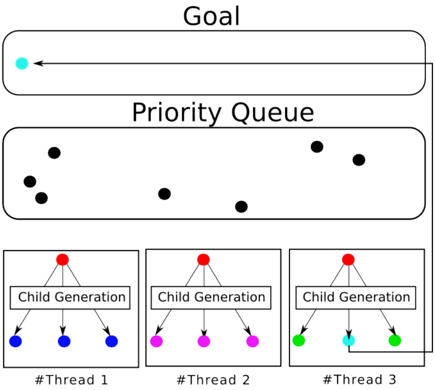
BOBPP, Priority Queue Principle



Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue

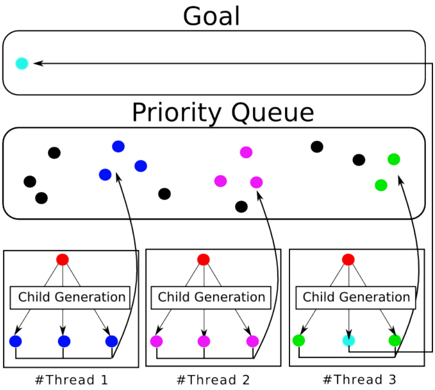
BOBPP, Priority Queue Principle



Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue

BOBPP, Priority Queue Principle



Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue

Outline

Protocol

Problems

- Quadratic Assignment Problem (QAP) (B&B algorithm)
- Golomb ruler Problem (D&C algorithm + OR-Tools Constraint Programming Solver)

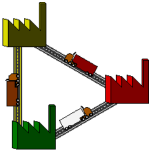


Figure: QAP

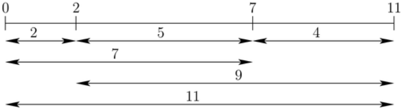
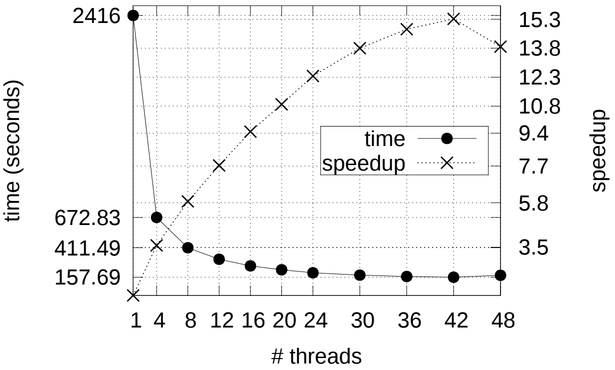


Figure: Golomb Ruler

Computers

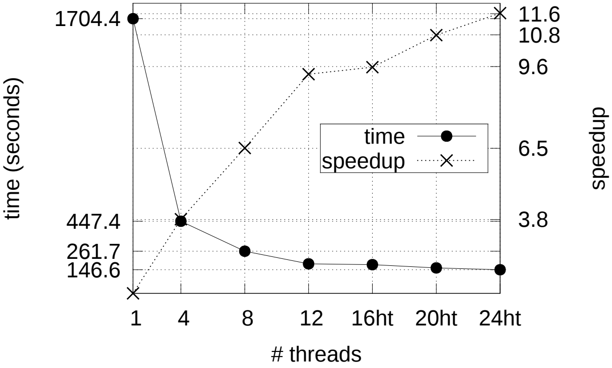
- Machine 1: AMD, 48 cores
- Machines 2 and 3: Intel, (with HT), 12 cores

BOBPP on shared memory machines, results



Mean computation time and speed-up for the QAP on Machine 1

BOBPP on shared memory machines, results



Mean computation time and speedup for the QAP on machine 2
Break on the curves beginning at 12 threads (Hyper-Threading)

Preliminary results using hybrid MPI/Pthreads

Machine	# MPI proc	# thr./ proc	total # thr.	time (s.)	speedup	% sended/ explored
1	4	12	48	85.46	19.95	0.265
1	7	7	49	68.93	24.73	0.290
2&3	2	12	24	137.38	12.41	0.178
2&3	4	6	24	133.42	12.78	0.226
all	16	6	96	**41.50**	**41.07**	0.455

Informations

- Speedup computed according to our best seq. time: 1,704.44 s.
- Very small nodes
- Mean number of nodes explored: 248,732,621
- Mean % of sended nodes compared to explored nodes: 0.256 %
- MPI/Threads version is more efficient than the threads one due to memory allocation contention.

Using a Constraint Programming Solver

OR-Tools

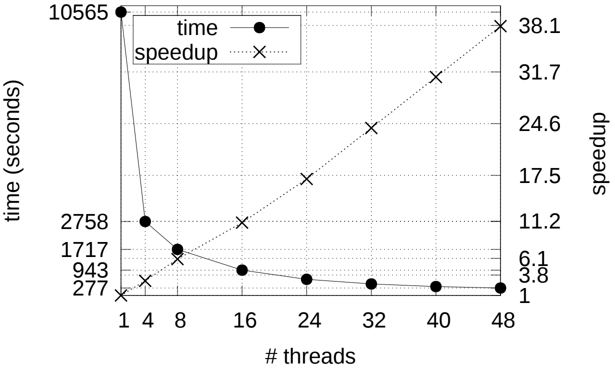
- Open source library and solver developed by Google
- Constraint programming methods we used are exact methods that handle a search tree.

Porting OR-Tools on top of BOBPP

- An OR-Tools solver handles its own search tree
- One OR-Tools solver per BOBPP search Algo
- Migration of sub-trees between OR-Tools solvers to perform load balancing

PAJERO project funded by OSEO a public-sector institution dedicated to economic development

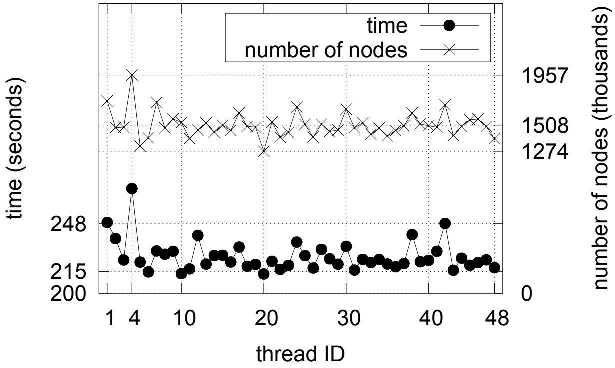
OR-Tools/BOBPP results



Execution time and speedup solving the Golomb of size 13

Good speedup

OR-Tools/BOBPP results



Load balancing on 48 threads solving the Golomb-13

Good load balancing

Outline

Conclusion

The BOBPP Framework manages the search-tree

- Whatever the final machine
- For a large variety of combinatorial problems

Advantages

- One algorithm developed by the user
- Easy to use
- Easy to test and develop better node generations
- Easy to find the best strategy

Perspectives

BOBPP Framework

- MPI/Threads need more experiments
- Perform test on larger machines (Grid5000)
- Reduce the memory allocation bottleneck using recursive search with a branch of preallocated nodes since a fixed depth.
- Perform tests with industrial problems developed by other members of the lab
  - Crew Scheduling: Branch& Price,
  - Power minimization of wireless sensor network: Branch& Cut,
  - Nurse planning: Constraint Programming
  - Restaurant waiters planning: Constraint Programming

End of presentation

Thank you

BOBPP sources

`http://forge.prism.uvsq.fr/projects/bobpp`