Work Stealing Strategies For Multi-Core Parallel Branch-and-Bound Algorithm Using Factorial Number System

R. Leroy[1], M. Mezmaz[2], N. Melab[1], D. Tuyttens[2] and M. Van Sevenant[2]

[1] INRIA/Université Lille1
[2] University of Mons

PMAM 2014, Orlando, February 15, 2014

Industrial and economic problems

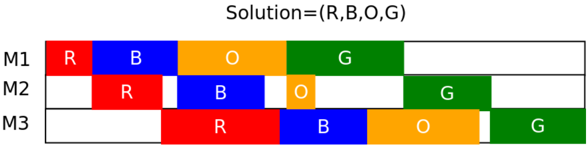
- E.g. : Logistics, telecommunications, IT, etc.
- Combinatorial optimization problems

Combinatorial optimization problems

- E.g. : flow-shop, TSP, QAP, etc.
- Many are permutation problems
- Often NP-hard problems
- Real instances are large in size

$\Rightarrow$ Many real problems are **permutation combinatorial optimization problems**

- N jobs to schedule on M machines
  - Each job has a processing time for each machine
- Constraints :
  - A machine can not be simultaneously assigned to two jobs
  - The order of the jobs is the same on all machines
- Makespan is the objective to minimize
  - The end date of the last job on the last machine



$\Rightarrow$ The flow-shop is a **permutation problem**

Approximative resolution methods

- Providing a good solution in a short time

Exact resolution methods

- The B&B is one of the most used algorithms
- Providing an optimal solution
- Using a huge computing power for real instances

Multi-core computing systems

- Since 2009, all sold desktop/notebook processors are multi-cores

$\Rightarrow$ Adapting B&B for multi-core computing
$\Rightarrow$ Finding an efficient work stealing strategy

- **Problem** and **solution**
  - $\boxed{/1234}$ is a problem (i.e. the root problem)
  - $\boxed{3142/}$ is a solution of the problem $\boxed{/1234}$
  - solutions($\boxed{/123}$)= { $\boxed{123/}$, $\boxed{132/}$, $\boxed{213/}$, $\boxed{231/}$, $\boxed{312/}$, $\boxed{321/}$ }
  - cost($\boxed{4132/}$)=50 $\Longleftrightarrow$ cost of the solution $\boxed{4132/}$ is 50
- $\boxed{2/134}$ is a **subproblem** of the problem $\boxed{/1234}$
  - solutions($\boxed{2/134}$) $\subset$ solutions($\boxed{/1234}$)
  - $\boxed{23/14}$ is a subproblem of the subproblem $\boxed{2/134}$
- { $\boxed{3/124}$, $\boxed{4/123}$, $\boxed{23/14}$, $\boxed{24/13}$ } is a **pool of subproblems**

$\Rightarrow$ The **B&B** uses three type of operands : **(sub)problem**, **pool of subproblems** and **solution**

- Selection operator
  - Pool of subproblems $\rightarrow$ Subproblem
  - depth-first({ $\boxed{3/124}$, $\boxed{4/123}$, $\boxed{23/14}$, $\boxed{24/13}$ })= $\boxed{23/14}$
- Bounding operator
  - Subproblem $\rightarrow$ Integer
  - lower($\boxed{2/134}$)$=$ 72 $\Longleftrightarrow$ $\forall$s$\in$solutions($\boxed{2/134}$)/cost(s)$\geq$ 72
- Branching operator
  - Subproblem $\rightarrow$ Pool of subproblems
  - { $\boxed{2/134}$ } $\Longleftrightarrow$ { $\boxed{21/34}$, $\boxed{23/14}$, $\boxed{24/13}$ }
- Elimination operator
  - (Subproblem,Integer) $\rightarrow$ Boolean
  - $\boxed{3142/}$ is the best found solution so far
  - lower($\boxed{2/134}$)$\geq$cost($\boxed{3142/}$) $\Rightarrow$ $\boxed{2/134}$ can be ignored

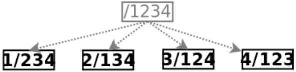
$\Rightarrow$ The **B&B** uses four operators : **selection**, **bounding**, **branching** and **elimination**

Operators	Selection Bounding Branching Elimination
Pool	/1234
Best	4132/ 50

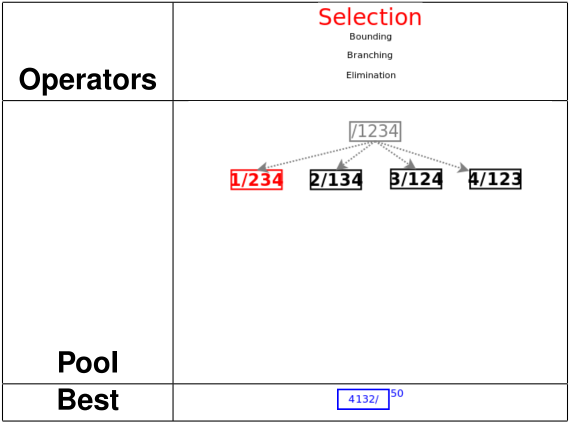
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	$\boxed{/1234}$
Best	$\boxed{4132/}$ 50

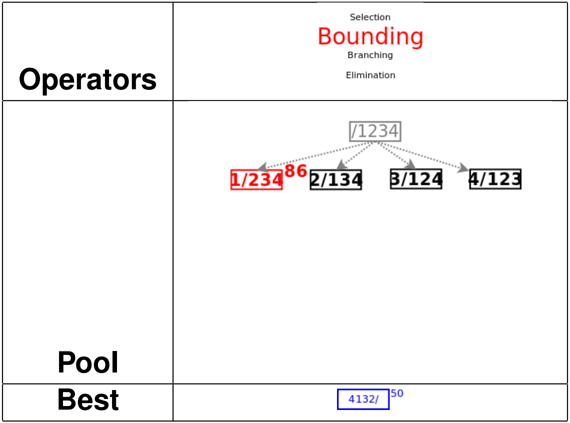
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding **Branching** Elimination
Pool	
Best	4132/ 50

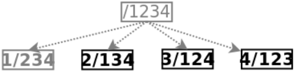
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**



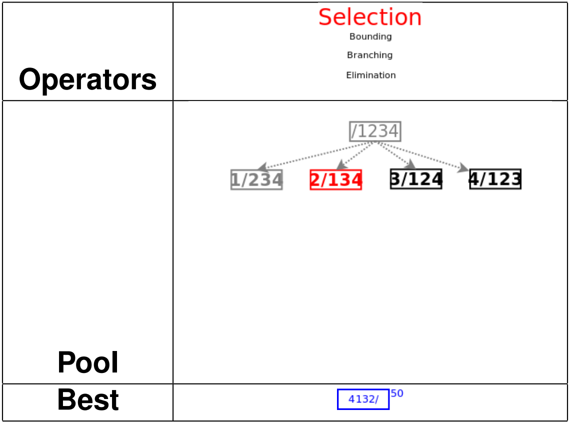
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**



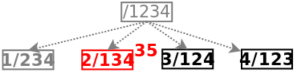
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	
Best	4132/ 50

⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**



⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

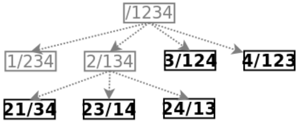
Operators	Selection Bounding Branching Elimination
Pool	
Best	4132/ 50

⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators

Selection
Bounding
Branching
Elimination

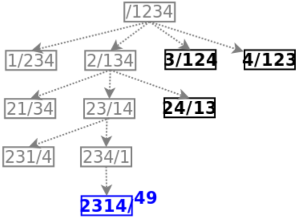
Pool



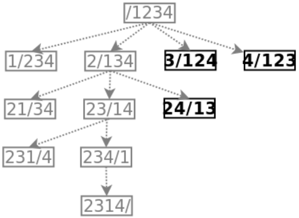
Best

4132/ 50

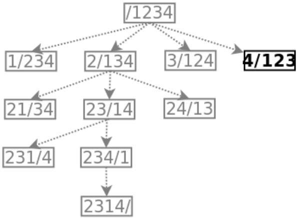
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	
Best	4132/ 50

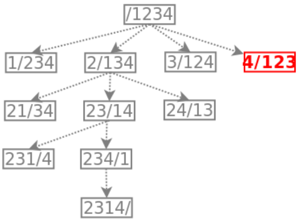
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection **Bounding** Branching Elimination
Pool	
Best	2314/ 49

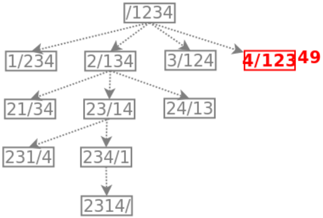
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	
Best	2314/ 49

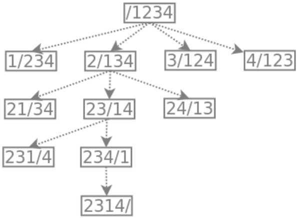
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	
Best	2314/ 49

⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection **Bounding** Branching Elimination
Pool	
Best	2314/ 49

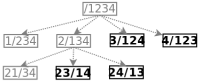
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

Operators	Selection Bounding Branching Elimination
Pool	
Best	2314/ 49

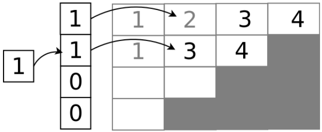
⇒ The **pool of subproblems** is a dynamically building **tree of subproblems**

$\Rightarrow$ Replace the **pool** by an **integer**, a **vector** and a **matrix**

- Each **cell of the matrix** $\Longleftrightarrow$ a **subproblem**
- The **last cell of the matrix** $\Longleftrightarrow$ to a **solution**

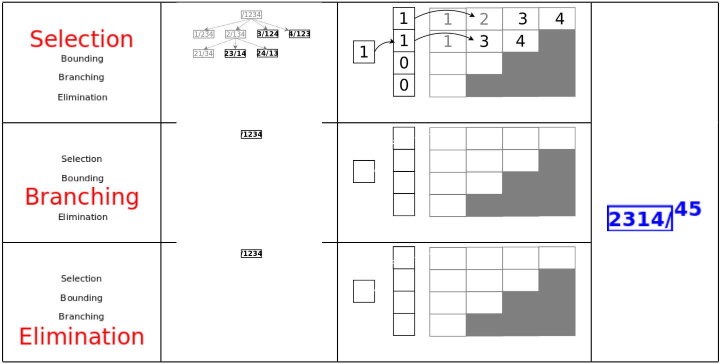


Pool of subproblems

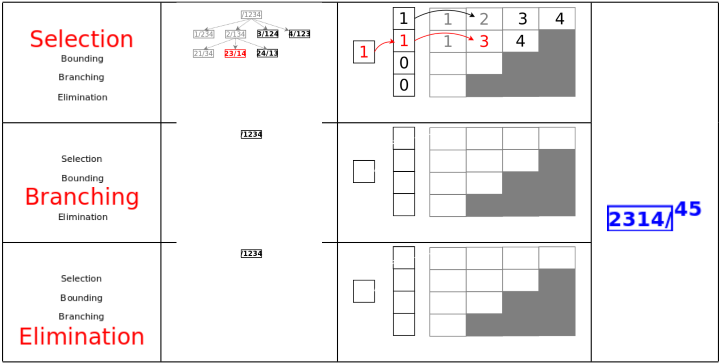


An integer, a vector and a matrix

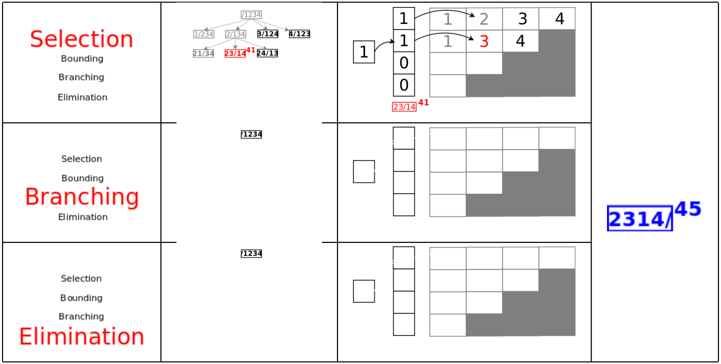
$\Rightarrow$ Some **B&B operators** must be adapted for the **new structure**



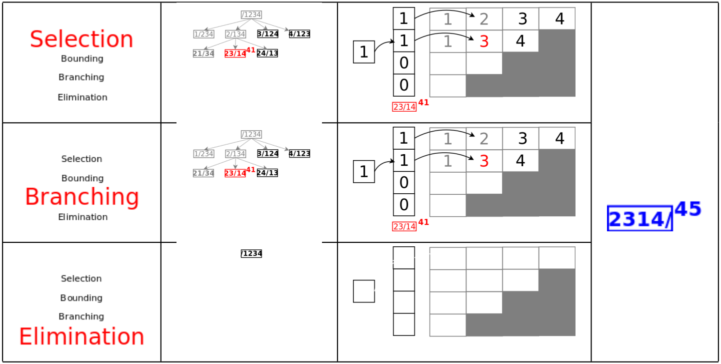
⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined



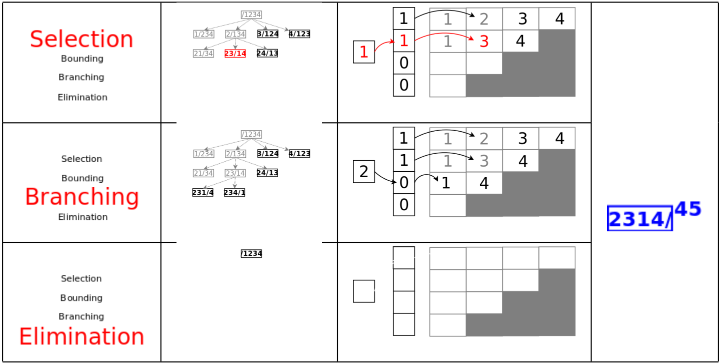
⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined



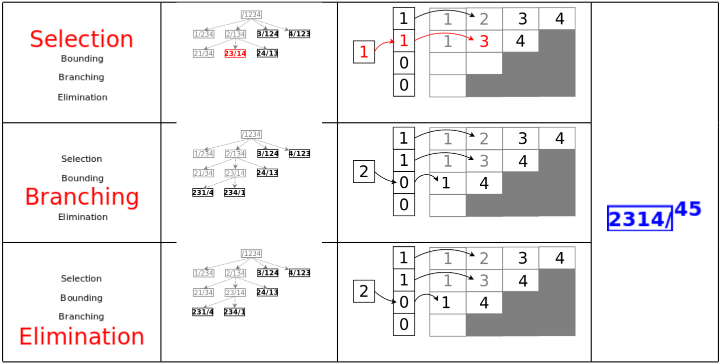
⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined



⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined

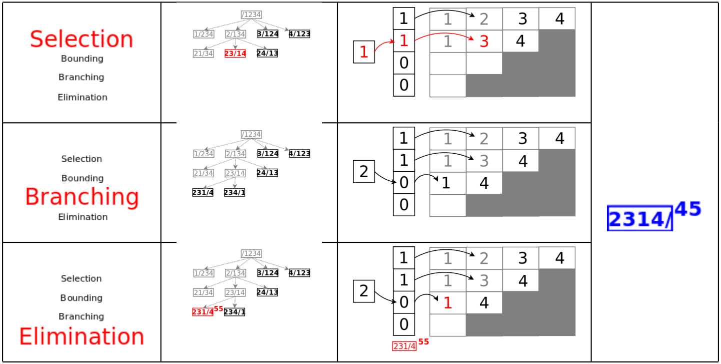


⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined

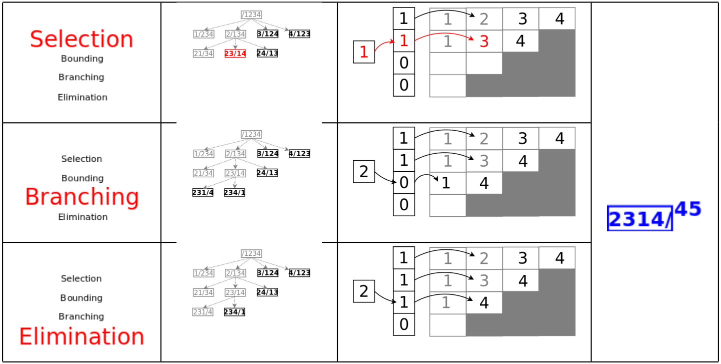


⇒ **Bounding** is the same

⇒ **Selection**, **branching** and **elimination** are redefined

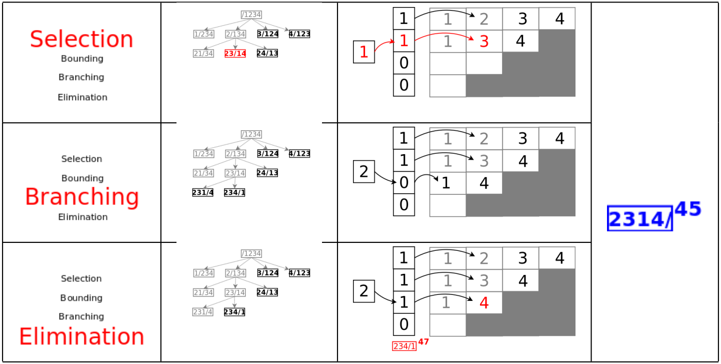


⇒ **Bounding** is the same
⇒ **Selection**, **branching** and **elimination** are redefined



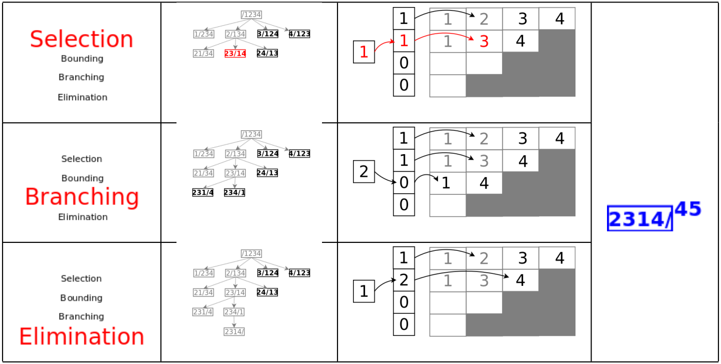
⇒ **Bounding** is the same

⇒ **Selection**, **branching** and **elimination** are redefined



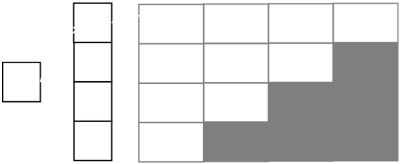
⇒ **Bounding** is the same

⇒ **Selection**, **branching** and **elimination** are redefined

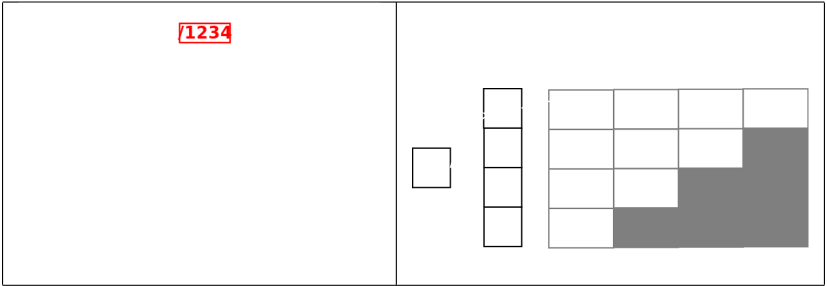


⇒ **Bounding** is the same

⇒ **Selection**, **branching** and **elimination** are redefined

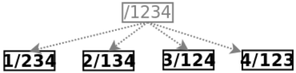
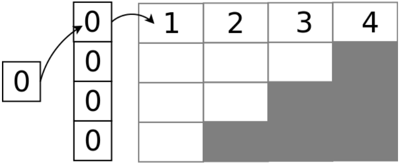
	
Pool-based B&B	Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

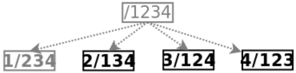


Pool-based B&B	Factoradic-based B&B

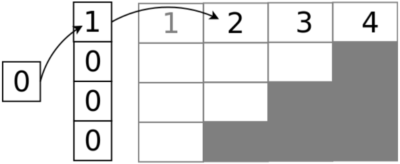
$\Rightarrow$ The **vector** behaves like a **factoradic counter**

	
Pool-based B&B	Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

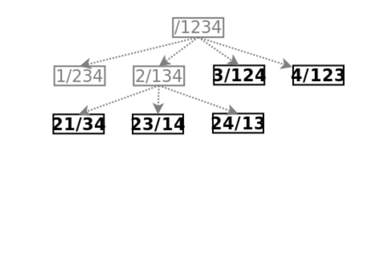


Pool-based B&B

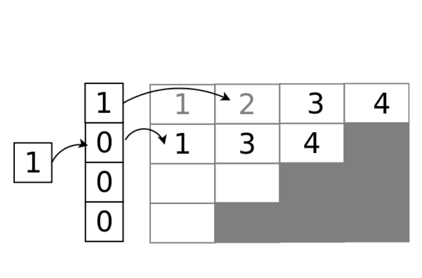


Factoradic-based B&B

⇒ The **vector** behaves like a **factoradic counter**

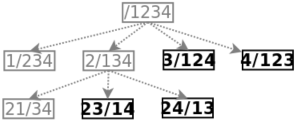


Pool-based B&B

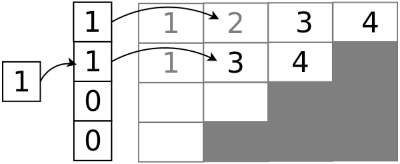


Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

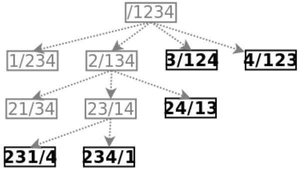
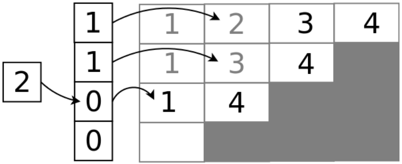


Pool-based B&B

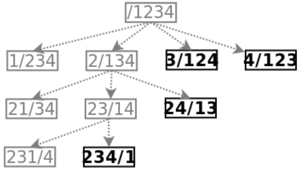
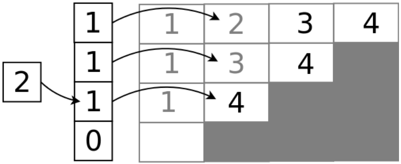


Factoradic-based B&B

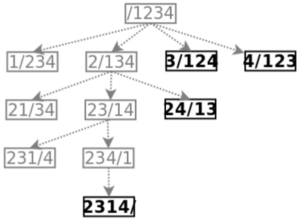
$\Rightarrow$ The **vector** behaves like a **factoradic counter**

	
Pool-based B&B	Factoradic-based B&B

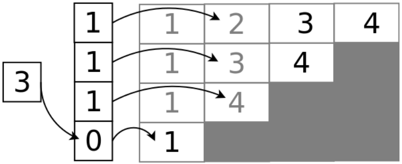
⇒ The **vector** behaves like a **factoradic counter**

	
Pool-based B&B	Factoradic-based B&B

⇒ The **vector** behaves like a **factoradic counter**

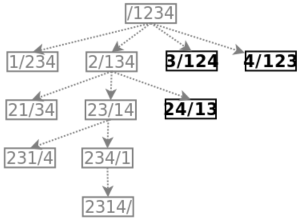


Pool-based B&B

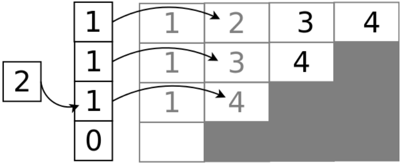


Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

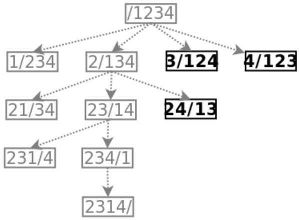


Pool-based B&B

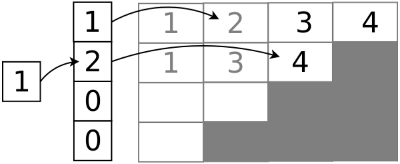


Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

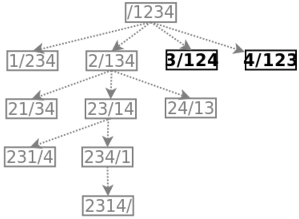
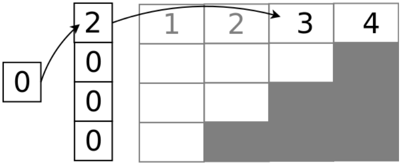


Pool-based B&B

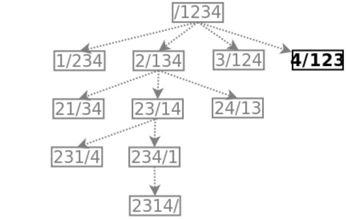


Factoradic-based B&B

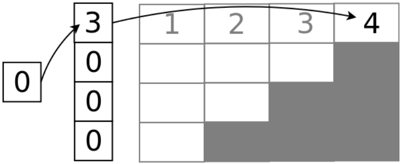
$\Rightarrow$ The **vector** behaves like a **factoradic counter**

	
Pool-based B&B	Factoradic-based B&B

⇒ The vector behaves like a factoradic counter

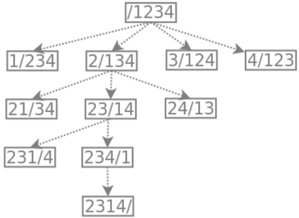
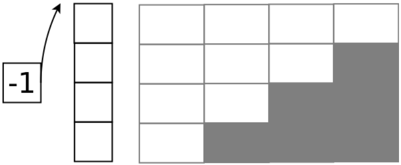


Pool-based B&B



Factoradic-based B&B

$\Rightarrow$ The **vector** behaves like a **factoradic counter**

	
Pool-based B&B	Factoradic-based B&B

⇒ The **vector** behaves like a **factoradic counter**

	Decimal number system	Factorial number system
Highest digit of the i^{th} position	9	i
Highest number with 5 digits	99999	43210
Weight of the i^{th} position	10^i	$i!$

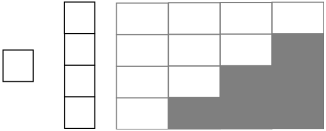
Factorial number system (also called factoradic)

- Was first used by [C-A. Laisant, 1888]
- Adapted to numbering permutations

$\Rightarrow$ With N jobs, the values of the **vector** belong to **[0,N ![**

$\Rightarrow$ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

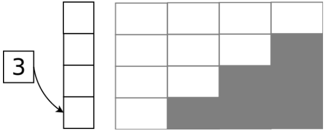
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



$\Rightarrow$ **Funfold** makes it possible to explore any **[A,B[**

$\Rightarrow$ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

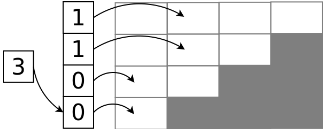
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



$\Rightarrow$ **Funfold** makes it possible to explore any **[A,B[**

⇒ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

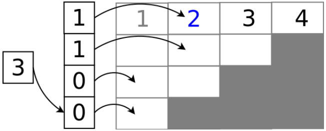
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



⇒ **Funfold** makes it possible to explore any **[A,B[**

$\Rightarrow$ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

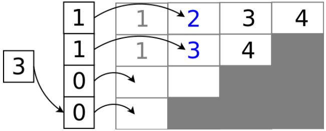
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



$\Rightarrow$ **Funfold** makes it possible to explore any **[A,B[**

$\Rightarrow$ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

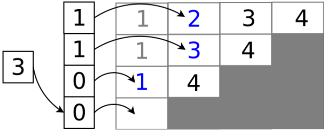
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



$\Rightarrow$ **Funfold** makes it possible to explore any **[A,B[**

⇒ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

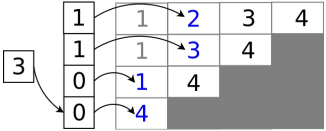
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



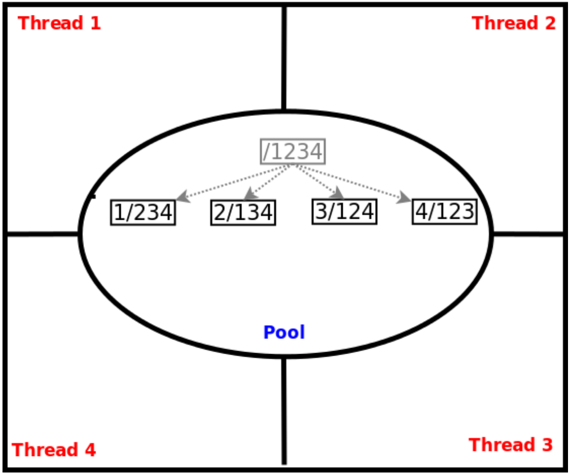
⇒ **Funfold** makes it possible to explore any **[A,B[**

$\Rightarrow$ Exploring any interval **[A,B[** instead of the whole **[0,N ![**

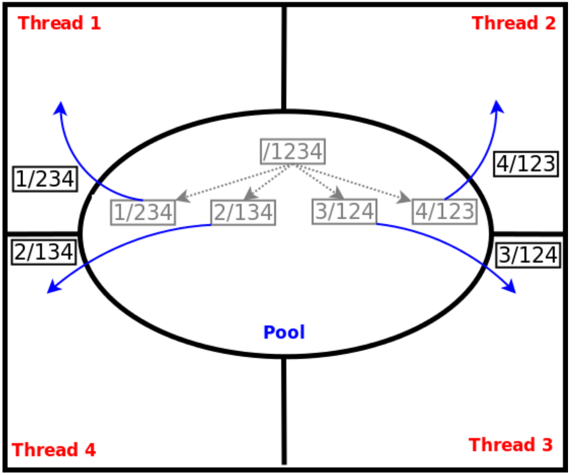
- A thread starts its exploration from **A**
- A thread stops when its **vector** is equal to **B**
- **Funfold operator**
  - Initialization of the **integer**, the **vector** and the **matrix**
- Example : **funfold([**$1100_{factoradic}$**,B[)**



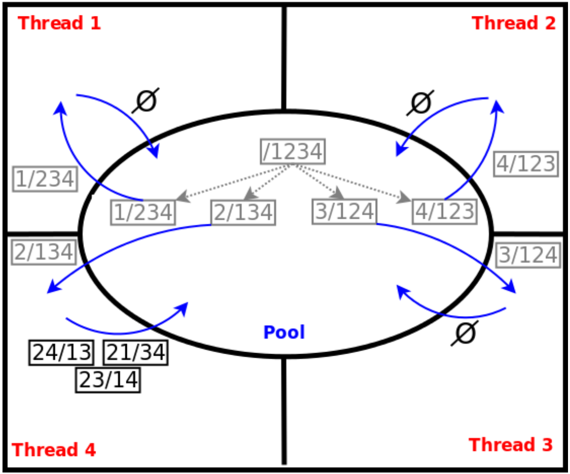
$\Rightarrow$ **Funfold** makes it possible to explore any **[A,B[**



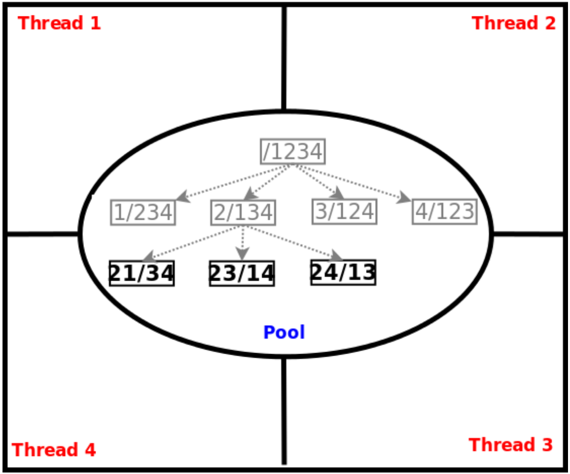
⇒ Work units are **subproblems**



⇒ Work units are subproblems

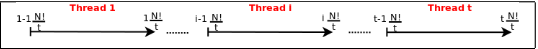


⇒ Work units are subproblems



$\Rightarrow$ **Work units** are **subproblems**

Initialization



j= C=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

⇒ **Work units** are **intervals of factoradics**
⇒ **4 factoradic-based strategies** are tested

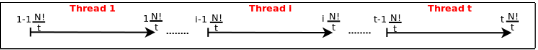
Initialization	

Before a work request	

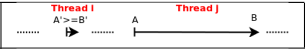
j= C=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

⇒ **Work units** are **intervals of factoradics**
⇒ **4 factoradic-based strategies** are tested

Initialization



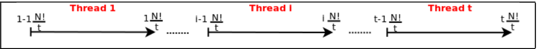
Before a work request



C= \ j=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

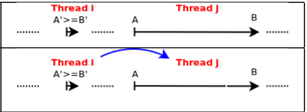
⇒ **Work units** are **intervals of factoradics**
⇒ **4 factoradic-based strategies** are tested

Initialization



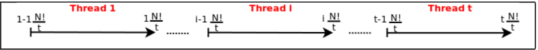
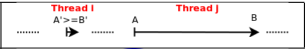
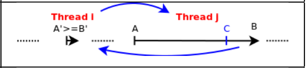
Before a work request

Work sharing



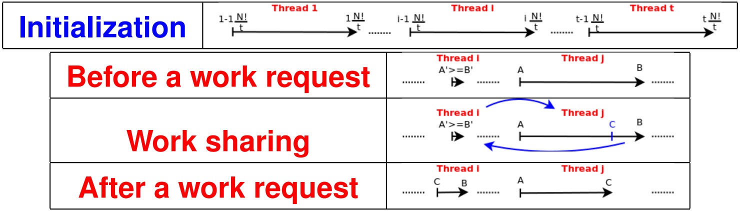
C= \ j=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

⇒ **Work units** are **intervals of factoradics**
⇒ **4 factoradic-based strategies** are tested

Initialization	
Before a work request	
Work sharing	

j= C=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

$\Rightarrow$ **Work units** are **intervals of factoradics**
$\Rightarrow$ **4 factoradic-based strategies** are tested



C= \ j=	t if (i=1) i-1 otherwise	random(1,t)	Thread with largest interval
(A+B)/2	Ring 1/2	Random	Largest interval
(A+B)/t	Ring 1/t	/	/

⇒ **Work units** are **intervals of factoradics**
⇒ **4 factoradic-based strategies** are tested

- Flow-shop instances
  - The 10 Taillard's instances with 20 machines and 20 jobs
  - The B&B is always initialized by the optimal solution
- Hardware and software testbed
  - 2 8-core Sandy Bridge E5-2670 processors
  - RedHat Linux distribution
- Time spent for managing the pool
  - Using the **clock_gettime** C function
  - Measuring time with a nanosecond precision

$\Rightarrow$ 16 threads are used to solve each instance

Instance	Time management : selection, branching, elimination and funfold (seconds)					Synchronization (seconds)				
	Pool	Factoradic				Pool	Factoradic			
		Ring 1/2	Ring 1/T	Random	Largest Interval		Ring 1/2	Ring 1/T	Random	Largest Interval
21	1292	181	137	139	142	40	11	5	5	6
22	719	100	72	72	74	23	8	2	3	2
23	4483	658	475	481	494	128	16	16	16	16
24	1355	172	128	127	131	42	12	5	5	5
25	1162	199	132	134	137	37	13	5	6	5
26	2278	299	222	224	229	76	14	9	9	9
27	2236	260	188	190	193	71	13	7	7	7
28	260	38	26	27	27	8	3	1	1	1
29	230	32	22	22	23	6	2	1	1	1
30	55	8	5	5	5	1	1	0.3	0.3	0.2
Average	**1407**	**195**	**141**	**142**	**146**	**43**	**9**	**5**	**5.7**	**5.6**

$\Rightarrow$ **Worst factoradic strategy** spends ~ 7.2 times less time than the **pool strategy**

Instances	Ring 1/2	Ring 1/T	Random	Largest Interval
21	263238	1321	640	392
22	189216	1161	808	393
23	980161	1194	926	441
24	296345	1631	978	377
25	428305	1433	929	487
26	439456	1140	880	406
27	445651	1037	974	369
28	85967	1595	603	331
29	75471	1215	883	420
30	21057	1105	562	321
Average	**322487**	**1283**	**818**	**393**

$\Rightarrow$ **Largest interval strategy** is better than the three other strategies

A conventional parallel **pool-based B&B** algorithm

- based on a **pool of subproblems**
- work units are **subproblems**

A new parallel **factoradic-based B&B** algorithm

- based on a new structure : **integer**, **vector** and **matrix**
- work units are **intervals of factoradics**

$\Rightarrow$ The **factoradic-based B&B** strategies outperform the **pool-based B&B** strategy
$\Rightarrow$ **Largest interval strategy** gives the best results

Generalization of this approach to other
$\Rightarrow$ Tree-based methods (B&C, B&P, etc.)
$\Rightarrow$ Types of optimization problems
$\Rightarrow$ Computing systems

Computing systems

- a **many-core** factoradic-based B&B for GPUs
- a **multi and many-core** factoradic-based B&B
- a **distributed multi and many-core** factoradic-based B&B

Distributed multi and many-core factoradic-based B&B

- **MPI** for the cluster level
- **OpenMP** for the multi-core processor level
- **CUDA** for the many-core processor level