Sort & Search techniques

V T'kindt, C Lenté

tkindt@univ-tours.fr, Université Francois-Rabelais, CNRS, Tours, France

March 2015

1. The principles of the original method
2. Formalization of the original method
3. Extension of Sort & Search
4. Application to a scheduling problem

Sort & Search : the principles

- It is an old technique which consists in **sorting** "data" to make the **search** for an optimal solution more efficient,
- It has been proposed by Horowitz and Sahni ([1]) to solve the knapsack problem,
- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- Assume we are given a decision/optimization problem (I denotes an instance of the problem),

[1] E. Horowitz and G. Sahni. Computing partitions with applications to the knapsack problem, Journal of the ACM, vol 21, pp.277-292, 1974

Sort & Search : the principles

- It is an old technique which consists in **sorting** "data" to make the **search** for an optimal solution more efficient,
- It has been proposed by Horowitz and Sahni ([1]) to solve the knapsack problem,
- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- Assume we are given a decision/optimization problem (I denotes an instance of the problem),

[1] E. Horowitz and G. Sahni. Computing partitions with applications to the knapsack problem, Journal of the ACM, vol 21, pp.277-292, 1974

Sort & Search : the principles

- It is an old technique which consists in **sorting** "data" to make the **search** for an optimal solution more efficient,
- It has been proposed by Horowitz and Sahni ([1]) to solve the knapsack problem,
- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- Assume we are given a decision/optimization problem (I denotes an instance of the problem),

[1] E. Horowitz and G. Sahni. Computing partitions with applications to the knapsack problem, Journal of the ACM, vol 21, pp.277-292, 1974

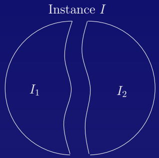
Sort & Search : the principles

- It is an old technique which consists in **sorting** “data” to make the **search** for an optimal solution more efficient,
- It has been proposed by Horowitz and Sahni ([1]) to solve the knapsack problem,
- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- Assume we are given a decision/optimization problem (I denotes an instance of the problem),

[1] E. Horowitz and G. Sahni. Computing partitions with applications to the knapsack problem, Journal of the ACM, vol 21, pp.277-292, 1974

Sort & Search : the principles

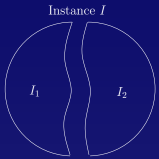
- The idea is the following : separate the instance into **2** sub-instances,



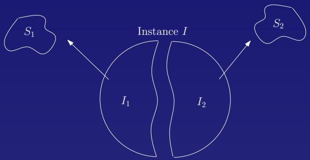
- Then, enumerate all partial solutions from I_1 and all partial solutions from I_2,

Sort & Search : the principles

- The idea is the following : separate the instance into **2** sub-instances,

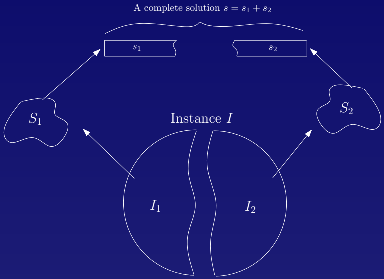


- Then, enumerate all partial solutions from I_1 and all partial solutions from I_2,



Sort & Search : the principles

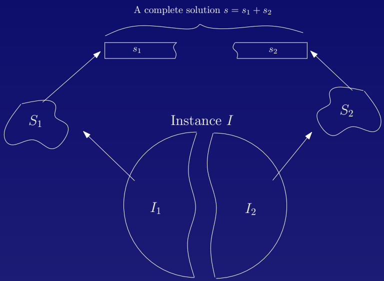
- By recombination of partial solutions, find the optimal solution of the initial problem



- The combinatoric appears when building S_1 and S_2 by enumeration (*sort* phase) and when finding in these sets the optimal solution (*search* phase).

Sort & Search : the principles

- By recombination of partial solutions, find the optimal solution of the initial problem



- The combinatoric appears when building S_1 and S_2 by enumeration (*sort* phase) and when finding in these sets the optimal solution (*search* phase).

Sort & Search : illustration

- Let us start with the KNAPSACK problem,
  - Let be $O = \{o_1, \ldots, o_n\}$ a set of n objects,
  - Each object o_i is defined by a value $v(o_i)$ and a weight $w(o_i)$, $1 \leq i \leq n$,
  - The, integer, capacity W of the knapsack.
  - Goal : Find $O' \subseteq O$ such that $\sum_{o \in O'} w(o) \leq W$ and $\sum_{o \in O'} v(o)$ is maximum.
- We can easily show that ENUM is in $O^*(2^n)$ time,

Sort & Search : illustration

- Let us start with the KNAPSACK problem,
  - Let be $O = \{o_1, \ldots, o_n\}$ a set of n objects,
  - Each object o_i is defined by a value $v(o_i)$ and a weight $w(o_i)$, $1 \leq i \leq n$,
  - The, integer, capacity W of the knapsack.
  - Goal : Find $O' \subseteq O$ such that $\sum_{o \in O'} w(o) \leq W$ and $\sum_{o \in O'} v(o)$ is maximum.
- We can easily show that ENUM is in $O^*(2^n)$ time,

Sort & Search : illustration

- We have $n = 6$, $O = \{a, b, c, d, e, f\}$ and $W = 9$.

O	a	b	c	d	e	f
v	3	4	2	5	1	3
w	4	2	1	3	2	5

$O_1 = \{a, b, c\}$ $\quad O_2 = \{d, e, f\}$

- Next, we enumerate the set of all possible assignments for O_1 (Table T_1),

T_1	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a, b\}$	$\{a, c\}$	$\{b, c\}$	$\{a, b, c\}$
$\sum v$	0	3	4	2	7	5	6	9
$\sum w$	0	4	2	1	6	5	3	7

Sort & Search : illustration

- We have $n = 6$, $O = \{a, b, c, d, e, f\}$ and $W = 9$.

O	a	b	c	d	e	f
v	3	4	2	5	1	3
w	4	2	1	3	2	5

$O_1 = \{a, b, c\}$ $\quad O_2 = \{d, e, f\}$

- Next, we enumerate the set of all possible assignments for O_1 (Table T_1),

T_1	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
$\sum v$	0	3	4	2	7	5	6	9
$\sum w$	0	4	2	1	6	5	3	7

Sort & Search : illustration

- We have $n = 6$, $O = \{a, b, c, d, e, f\}$ and $W = 9$.

O	a	b	c	d	e	f
v	3	4	2	5	1	3
w	4	2	1	3	2	5

$O_1 = \{a, b, c\}$ $O_2 = \{d, e, f\}$

- Next, we enumerate the set of all possible assignments for O_1 (Table T_1),

T_1	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
$\sum v$	0	3	4	2	7	5	6	9
$\sum w$	0	4	2	1	6	5	3	7

Sort & Search : illustration

- We have $n = 6$, $O = \{a, b, c, d, e, f\}$ and $W = 9$.

O	a	b	c	d	e	f
v	3	4	2	5	1	3
w	4	2	1	3	2	5

$O_1 = \{a, b, c\}$ $\quad O_2 = \{d, e, f\}$

- Next, we enumerate the set of all possible assignments for O_1 (Table T_1),

T_1	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a, b\}$	$\{a, c\}$	$\{b, c\}$	$\{a, b, c\}$
$\sum v$	0	3	4	2	7	5	6	9
$\sum w$	0	4	2	1	6	5	3	7

Sort & Search : illustration

- Next, we do the same for O_2 (Table T_2),

T_2	$\emptyset$	$\{e\}$	$\{d\}$	$\{f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d,e,f\}$
$\sum v$	0	1	5	3	6	4	8	9
$\sum w$	0	2	3	5	5	7	8	10
ℓ_k	1	2	3	3	5	5	7	8

Note : In table T_2, columns are sorted by increasing order of $\sum w$.
Note : ℓ_k is the column number with maximum $\sum v$ "on the left" of the current column.

- That was the *Sort* phase !
- Running time (and space) should be "about" $2^{n/2}$,

Sort & Search : illustration

- Next, we do the same for O_2 (Table T_2),

T_2	$\emptyset$	$\{e\}$	$\{d\}$	$\{f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d,e,f\}$
$\sum v$	0	1	5	3	6	4	8	9
$\sum w$	0	2	3	5	5	7	8	10
ℓ_k	1	2	3	3	5	5	7	8

  Note : In table T_2, columns are sorted by increasing order of $\sum w$.
  Note : ℓ_k is the column number with maximum $\sum v$ "on the left" of the current column.

- That was the *Sort* phase !
- Running time (and space) should be "about" $2^{n/2}$,

Sort & Search : illustration

- Next, we do the same for O_2 (Table T_2),

T_2	$\emptyset$	$\{e\}$	$\{d\}$	$\{f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d,e,f\}$
$\sum v$	0	1	5	3	6	4	8	9
$\sum w$	0	2	3	5	5	7	8	10
ℓ_k	1	2	3	3	5	5	7	8

  Note : In table T_2, columns are sorted by increasing order of $\sum w$.
  Note : ℓ_k is the column number with maximum $\sum v$ "on the left" of the current column.
- That was the *Sort* phase !
- Running time (and space) should be "about" $2^{n/2}$,

Sort & Search : illustration

- Search phase can start,
- For any column $j \in T_1$, find the "best" complementing column $k \in T_2$,
- Best : column k which maximizes $\sum w$... then column ℓ_k will be the one which maximizes $\sum v$,
- The search phase leads to,

j	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
k	$\{d,f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d\}$	$\{d\}$	$\{d,e\}$	$\{e\}$
$w(O'_j)+w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j)+v(O'_{\ell_k})$	8	9	10	10	12	10	12	10

Consequently, the optimal solution has value 12 and is achieved with $\{a,b,d\}$ or $\{b,c,d,e\}$.

Sort & Search : illustration

- Search phase can start,
- For any column $j \in T_1$, find the “best” complementing column $k \in T_2$,
- Best : column k which maximizes $\sum w$... then column ℓ_k will be the one which maximizes $\sum v$,
- The search phase leads to,

j	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
k	$\{d,f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d\}$	$\{d\}$	$\{d,e\}$	$\{e\}$
$w(O'_j)+w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j)+v(O'_{\ell_k})$	8	9	10	10	12	10	12	10

Consequently, the optimal solution has value 12 and is achieved with $\{a,b,d\}$ or $\{b,c,d,e\}$.

Sort & Search : illustration

- Search phase can start,
- For any column $j \in T_1$, find the "best" complementing column $k \in T_2$,
- Best : column k which maximizes $\sum w$... then column ℓ_k will be the one which maximizes $\sum v$,
- The search phase leads to,

j	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
k	$\{d,f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d\}$	$\{d\}$	$\{d,e\}$	$\{e\}$
$w(O'_j)+w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j)+v(O'_{\ell_k})$	8	9	10	10	12	10	12	10

Consequently, the optimal solution has value 12 and is achieved with $\{a,b,d\}$ or $\{b,c,d,e\}$.

Sort & Search : illustration

- Search phase can start,
- For any column $j \in T_1$, find the "best" complementing column $k \in T_2$,
- Best : column k which maximizes $\sum w$... then column ℓ_k will be the one which maximizes $\sum v$,
- The search phase leads to,

j	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
k	$\{d,f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d\}$	$\{d\}$	$\{d,e\}$	$\{e\}$
$w(O'_j)+w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j)+v(O'_{\ell_k})$	8	9	10	10	12	10	12	10

Consequently, the optimal solution has value 12 and is achieved with $\{a, b, d\}$ or $\{b, c, d, e\}$.

Sort & Search : formalization

- *Sort & Search* is a powerfull technique which can be applied to a lot of problems,
- Intuitively, to be applicable efficiently, problems must satisfy two properties :
  - Two partial solutions can be combined in polynomial time to get a feasible solution,
  - The Sort phase must enable to lead to a Search phase which [illegible] tables.
- *Sort & Search*, as introduced by Horowitz and Sahni, can be applied to a class of problems called *Single Constraint Problems* (SCP),

Sort & Search : formalization

- *Sort & Search* is a powerfull technique which can be applied to a lot of problems,
- Intuitively, to be applicable efficiently, problems must satisfy two properties :
  1. Two partial solutions can be combined in polynomial time to get a feasible solution,
  2. The Sort phase must enable to lead to a Search phase which complexity does not exceed the one required to build the tables.
- *Sort & Search*, as introduced by Horowitz and Sahni, can be applied to a class of problems called *Single Constraint Problems* (SCP),

Sort & Search : formalization

- *Sort & Search* is a powerfull technique which can be applied to a lot of problems,
- Intuitively, to be applicable efficiently, problems must satisfy two properties :
  1. Two partial solutions can be combined in polynomial time to get a feasible solution,
  2. The Sort phase must enable to lead to a Search phase which complexity does not exceed the one required to build the tables.
- *Sort & Search*, as introduced by Horowitz and Sahni, can be applied to a class of problems called *Single Constraint Problems* (SCP),

Sort & Search : formalization

- *Sort & Search* is a powerfull technique which can be applied to a lot of problems,
- Intuitively, to be applicable efficiently, problems must satisfy two properties :
  1. Two partial solutions can be combined in polynomial time to get a feasible solution,
  2. The Sort phase must enable to lead to a Search phase which complexity does not exceed the one required to build the tables.
- *Sort & Search*, as introduced by Horowitz and Sahni, can be applied to a class of problems called *Single Constraint Problems* (SCP),

Sort & Search : formalization

- *Sort & Search* is a powerfull technique which can be applied to a lot of problems,
- Intuitively, to be applicable efficiently, problems must satisfy two properties :
  1. Two partial solutions can be combined in polynomial time to get a feasible solution,
  2. The Sort phase must enable to lead to a Search phase which complexity does not exceed the one required to build the tables.
- *Sort & Search*, as introduced by Horowitz and Sahni, can be applied to a class of problems called *Single Constraint Problems* (SCP),

Sort & Search : formalization

- Let be $A = (\vec{a}_1,\ \vec{a}_2, \dots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = ((b_1, b'_1), (b_2, b'_2) \dots\ (b_{n_B}, b'_{n_B}))$ a table of n_B couples,
- Let f and g' be two functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$, increasing with respect to their last variable,
- The (SCP) :

$$\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k)\\
&\text{s.t.}\\
&\quad g'(\vec{a}_j, b'_k) \geq 0\\
&\quad \vec{a}_j \in A,\ (b_k, b'_k) \in B.
\end{aligned}$$

- **There exists a *Sort & Search* algorithm in $O(n_B \log_2(n_B) + n_A \log_2(n_B))$ time and $O(n_A + n_B)$ space.**
- KNAPSACK : $n_A = n_B = 2^{\frac{n}{2}} \Rightarrow O^*(2^{\frac{n}{2}})$ time and space.

Sort & Search : formalization

- Let be $A = (\vec{a}_1,\ \vec{a}_2, \ldots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = ((b_1, b'_1), (b_2, b'_2) \ldots\ (b_{n_B}, b'_{n_B}))$ a table of n_B couples,
- Let f and g' be two functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$, increasing with respect to their last variable,
- The (SCP) :

$$\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k)\\
&\text{s.t.}\\
&\qquad g'(\vec{a}_j, b'_k) \geq 0\\
&\qquad \vec{a}_j \in A,\ (b_k, b'_k) \in B.
\end{aligned}$$

- **There exists a *Sort & Search* algorithm in $O(n_B \log_2(n_B) + n_A \log_2(n_B))$ time and $O(n_A + n_B)$ space.**
- KNAPSACK : $n_A = n_B = 2^{\frac{n}{2}} \Rightarrow O^*(2^{\frac{n}{2}})$ time and space.

Sort & Search : formalization

- Let be $A = (\vec{a}_1,\ \vec{a}_2, \ldots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = ((b_1, b'_1),\ (b_2, b'_2) \ldots\ (b_{n_B}, b'_{n_B}))$ a table of n_B couples,
- Let f and g' be two functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$, increasing with respect to their last variable,
- The (SCP) :

$$\begin{aligned}&\text{Minimize } f(\vec{a}_j, b_k)\\ &\text{s.t.}\\ &\quad g'(\vec{a}_j, b'_k) \geq 0\\ &\quad \vec{a}_j \in A,\ (b_k, b'_k) \in B.\end{aligned}$$

- **There exists a *Sort & Search* algorithm in $O(n_B \log_2(n_B) + n_A \log_2(n_B))$ time and $O(n_A + n_B)$ space.**
- KNAPSACK : $n_A = n_B = 2^{\frac{n}{2}} \Rightarrow O^*(2^{\frac{n}{2}})$ time and space.

Sort & Search : generalization

- We can extend the original *Sort & Search* approach to *Multiple Constraint Problems* (MCP),
- Let be $A = (\vec{a}_1,\ \vec{a}_2, \ldots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = (\vec{b}_1,\ \vec{b}_2, \ldots \vec{b}_{n_B})$ a table of n_B vectors $\vec{b}_k = (b_k^0, b_k^1, \ldots, b_k^{d_B})$ of dimension $d_B + 1$,
- Let f and g_ℓ $(1 \leq \ell \leq d_B)$ be $d_B + 1$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ (increasing with respect to their last variable),
- The (MCP) is defined by :

$$
\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k^0) \\
&\text{s.t.} \\
&\quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B) \\
&\quad \vec{a}_j \in A,\ \vec{b}_k \in B.
\end{aligned}
$$

Sort & Search : generalization

- We can extend the original *Sort & Search* approach to *Multiple Constraint Problems* (MCP),
- Let be $A = (\vec{a}_1,\ \vec{a}_2, \dots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = (\vec{b}_1,\ \vec{b}_2, \dots \vec{b}_{n_B})$ a table of n_B vectors $\vec{b}_k = (b_k^0, b_k^1, \dots, b_k^{d_B})$ of dimension $d_B + 1$,
- Let f and g_ℓ $(1 \leq \ell \leq d_B)$ be $d_B + 1$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ (increasing with respect to their last variable),
- The (MCP) is defined by :

$$
\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k^0) \\
&\text{s.t.} \\
&\quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B) \\
&\quad \vec{a}_j \in A,\ \vec{b}_k \in B.
\end{aligned}
$$

Sort & Search : generalization

- We can extend the original *Sort & Search* approach to *Multiple Constraint Problems* (MCP),
- Let be $A = (\vec{a}_1,\ \vec{a}_2, \dots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = (\vec{b}_1,\ \vec{b}_2, \dots \vec{b}_{n_B})$ a table of n_B vectors $\vec{b}_k = (b_k^0, b_k^1, \dots, b_k^{d_B})$ of dimension $d_B + 1$,
- Let f and g_ℓ $(1 \leq \ell \leq d_B)$ be $d_B + 1$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ (increasing with respect to their last variable),
- The (MCP) is defined by :

$$
\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k^0)\\
&\text{s.t.}\\
&\qquad g_\ell(\vec{a}_j, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)\\
&\qquad \vec{a}_j \in A,\ \vec{b}_k \in B.
\end{aligned}
$$

Sort & Search : generalization

- We can extend the original *Sort & Search* approach to *Multiple Constraint Problems* (MCP),
- Let be $A = (\vec{a}_1,\ \vec{a}_2, \ldots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = (\vec{b}_1,\ \vec{b}_2, \ldots \vec{b}_{n_B})$ a table of n_B vectors $\vec{b}_k = (b_k^0, b_k^1, \ldots, b_k^{d_B})$ of dimension $d_B + 1$,
- Let f and g_ℓ $(1 \leq \ell \leq d_B)$ be $d_B + 1$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ (increasing with respect to their last variable),
- The (MCP) is defined by :

Minimize $f(\vec{a_j}, b_k^0)$

s.t.

$$g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$$
$$\vec{a_j} \in A,\ \vec{b_k} \in B.$$

Sort & Search : generalization

- We can extend the original *Sort & Search* approach to *Multiple Constraint Problems* (MCP),
- Let be $A = (\vec{a}_1,\ \vec{a}_2, \ldots \vec{a}_{n_A})$ a table of n_A vectors of dimension d_A,
- Let be $B = (\vec{b}_1,\ \vec{b}_2, \ldots \vec{b}_{n_B})$ a table of n_B vectors $\vec{b}_k = (b_k^0, b_k^1, \ldots, b_k^{d_B})$ of dimension $d_B + 1$,
- Let f and g_ℓ $(1 \leq \ell \leq d_B)$ be $d_B + 1$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ (increasing with respect to their last variable),
- The (MCP) is defined by :

$$\begin{aligned}
&\text{Minimize } f(\vec{a}_j, b_k^0) \\
&\text{s.t.} \\
&\quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B) \\
&\quad \vec{a}_j \in A,\ \vec{b}_k \in B.
\end{aligned}$$

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $Q = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $Q = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $Q = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $Q = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $Q = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- The same process, as in the (SCP), will be iterated : for each vector $\vec{a_j} \in A$, find the vector $\vec{b_k}$ answering the constraints and minimizing f,
- Assume that $\vec{a_j}$ is given,
- For any constraint $g_\ell(\vec{a_j}, b_k^\ell) \geq 0, \quad (1 \leq \ell \leq d_B)$...
- ... let be $\beta_j^k = \min\{b_k^\ell | 1 \leq k \leq n_B$ and $g_\ell(\vec{a_j}, b_k^\ell) \geq 0\}$
- Beside, let be $\beta \in \{b_k^0 | 1 \leq k \leq n_B\}$,
- If there is at least one vector $\vec{b_k} \in B$ with coordinates in $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$... then we know that the optimal solution of the (MCP) (when $\vec{a_j}$ is fixed) is at most β,
- We can iterate through all values of β,

Sort & Search : generalization

- Next question : how computing efficiently $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$?
- This is a *rectangular query*,
- We can make use of range trees to sort the vectors $\vec{b_k} \in B$,

Sort & Search : generalization

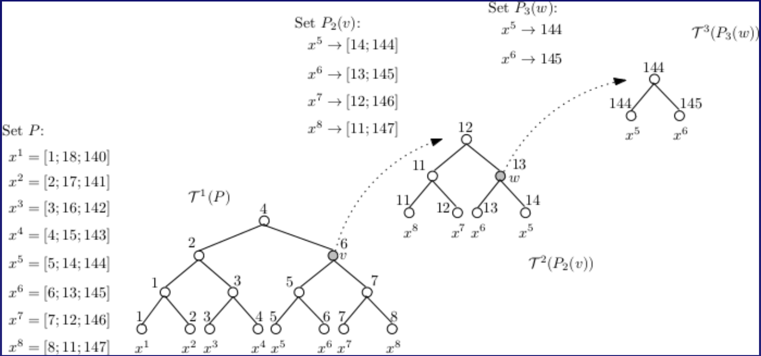
- Next question : how computing efficiently $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$?
- This is a *rectangular query*,
- We can make use of range trees to sort the vectors $\vec{b_k} \in B$,

Sort & Search : generalization

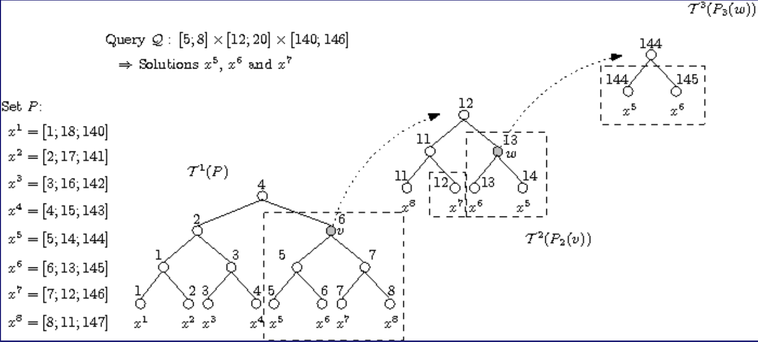
- Next question : how computing efficiently $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$?
- This is a *rectangular query*,
- We can make use of range trees to sort the vectors $\vec{b_k} \in B$,

Sort & Search : generalization

- Next question : how computing efficiently $\mathcal{Q} = [-\infty; \beta] \times [\beta_j^1; +\infty] \times ... \times [\beta_j^{d_B}; +\infty]$?
- This is a *rectangular query*,
- We can make use of range trees to sort the vectors $\vec{b_k} \in B$,



Sort & Search : generalization



Sort & Search : generalization

- What about the complexity ?
- ... we can establish a *Sort & Search* algorithm in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time and $O(n_B \log_2^{d_B-1}(n_B))$ space ([7]).

[7] C. Lente, M. Liedloff, A. Soukhal and V. T'kindt. On an extension of the Sort & Search method with application to scheduling theory, Theoretical Computer Science, vol 511, pp. 13-22, 2013.

Sort & Search : generalization

- What about the complexity ?
- ... we can establish a *Sort & Search* algorithm in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time and $O(n_B \log_2^{d_B-1}(n_B))$ space ([7]).

[7] C. Lente, M. Liedloff, A. Soukhal and V. T'kindt. On an extension of the Sort & Search method with application to scheduling theory, Theoretical Computer Science, vol 511, pp. 13-22, 2013.

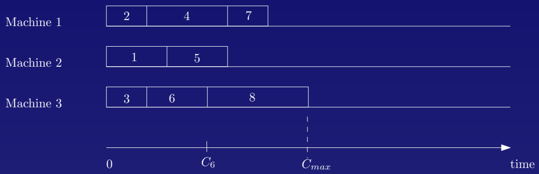
Sort & Search : an application

- Consider the following scheduling problem :
  - 3 identical machines are available to process n jobs,
  - Each job i is defined by a processing time p_i and can be processed by any of the 3 machines,
  - Find a schedule which minimizes the makespan $C_{max} = \max_i(C_i)$ with C_i the completion time of job i.

- This problem is $\mathcal{NP}$-hard.
- The worst-case time complexity of ENUM is in $O^*(3^n)$,

Sort & Search : an application

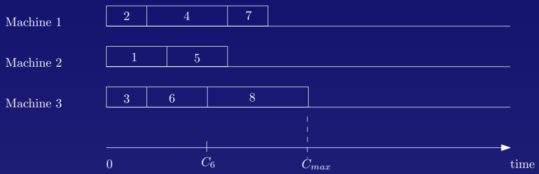
- Consider the following scheduling problem :
  - 3 identical machines are available to process n jobs,
  - Each job i is defined by a processing time p_i and can be processed by any of the 3 machines,
  - Find a schedule which minimizes the makespan $C_{max} = \max_i(C_i)$ with C_i the completion time of job i.



- This problem is $\mathcal{NP}$-hard.
- The worst-case time complexity of ENUM is in $O^*(3^n)$,

Sort & Search : an application

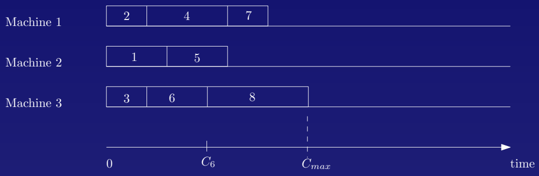
- Consider the following scheduling problem :
  - 3 identical machines are available to process n jobs,
  - Each job i is defined by a processing time p_i and can be processed by any of the 3 machines,
  - Find a schedule which minimizes the makespan $C_{max} = \max_i(C_i)$ with C_i the completion time of job i.



- This problem is $\mathcal{NP}$-hard.
- The worst-case time complexity of ENUM is in $O^*(3^n)$,

Sort & Search : an application

- Consider the following scheduling problem :
  - 3 identical machines are available to process n jobs,
  - Each job i is defined by a processing time p_i and can be processed by any of the 3 machines,
  - Find a schedule which minimizes the makespan $C_{max} = \max_i(C_i)$ with C_i the completion time of job i.



- This problem is $\mathcal{NP}$-hard.
- The worst-case time complexity of `ENUM` is in $O^*(3^n)$,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

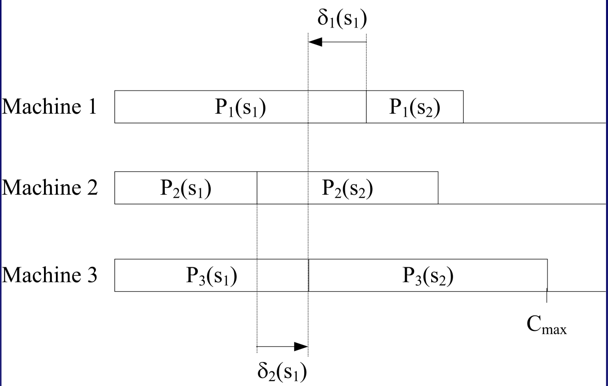
Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)

- Let I be an instance with n jobs given in a set $\mathcal{J}$,
- Let $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ be the subset of the $\lfloor \frac{n}{2} \rfloor$ first job of $\mathcal{J}$,
- Let $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ be the subset of the $\lceil \frac{n}{2} \rceil$ last job of $\mathcal{J}$,
- Let be $\mathcal{E}_1^j = (E_{1,1}^j, E_{1,2}^j, E_{1,3}^j)$ a 3-partition of I_1 $(1 \leq j \leq 3^{|I_1|})$,
- We associate to it a schedule s_1^j containing the sequence of jobs on machines,
- Similarly, let be $\mathcal{E}_2^k$ a 3-partition of I_2 $(1 \leq j \leq 3^{|I_2|})$,
- We associate to it a schedule s_2^k containing the sequence of jobs on machines,

Sort & Search : an application (main lines)



Sort & Search : an application (main lines)

$$\left\{\begin{array}{ll}\vec{a}_j = & (\delta_1(s_1^j), \delta_2(s_1^j)) \\ (b_k^0, b_k^1, b_k^2) = & (\delta_1(s_2^k) + \delta_2(s_2^k), \delta_1(s_2^k), \delta_2(s_2^k)) \\ f(\vec{a}_j, b_k^0) = & (P + \delta_1(s_1^j) + \delta_2(s_1^j) + \delta_1(s_2^k) + \delta_2(s_2^k))/3 \\ g_1(\vec{a}_j, b_k^1) = & \delta_1(s_1^j) + \delta_1(s_2^k) \\ g_2(\vec{a}_j, b_k^2) = & \delta_2(s_1^j) + \delta_2(s_2^k) \end{array}\right. \quad (1)$$

Besides f, g_1 are g_2 increasing function with respect to their last variable.

Sort & Search : an application

- The complexity of *Sort & Search* is in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time,
- Starting from I_1 and I_2, tables A and B have respectively $n_A = n_B = 3^{\frac{n}{2}}$ columns,
- Besides, $d_A = 2$, and $d_B = 2$
- Then, the worst-case time complexity is in $O(3^{\frac{n}{2}} \log_2^2(3^{\frac{n}{2}}) + 3^{\frac{n}{2}} \log_2^4(3^{\frac{n}{2}})) = O^*(3^{\frac{n}{2}}) \approx O^*(1.7321^n)$.

Sort & Search : an application

- The complexity of *Sort & Search* is in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time,
- Starting from I_1 and I_2, tables A and B have respectively $n_A = n_B = 3^{\frac{n}{2}}$ columns,
- Besides, $d_A = 2$, and $d_B = 2$
- Then, the worst-case time complexity is in $O(3^{\frac{n}{2}} \log_2^2(3^{\frac{n}{2}}) + 3^{\frac{n}{2}} \log_2^4(3^{\frac{n}{2}})) = O^*(3^{\frac{n}{2}}) \approx O^*(1.7321^n)$.

Sort & Search : an application

- The complexity of *Sort & Search* is in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time,
- Starting from I_1 and I_2, tables A and B have respectively $n_A = n_B = 3^{\frac{n}{2}}$ columns,
- Besides, $d_A = 2$, and $d_B = 2$
- Then, the worst-case time complexity is in $O(3^{\frac{n}{2}} \log_2^2(3^{\frac{n}{2}}) + 3^{\frac{n}{2}} \log_2^4(3^{\frac{n}{2}})) = O^*(3^{\frac{n}{2}}) \approx O^*(1.7321^n)$.

Sort & Search : an application

- The complexity of *Sort & Search* is in $O(n_B \log_2^{d_B}(n_B) + n_A \log_2^{d_B+2}(n_B))$ time,
- Starting from I_1 and I_2, tables A and B have respectively $n_A = n_B = 3^{\frac{n}{2}}$ columns,
- Besides, $d_A = 2$, and $d_B = 2$
- Then, the worst-case time complexity is in $O(3^{\frac{n}{2}} \log_2^2(3^{\frac{n}{2}}) + 3^{\frac{n}{2}} \log_2^4(3^{\frac{n}{2}})) = O^*(3^{\frac{n}{2}}) \approx O^*(1.7321^n)$.

Conclusions

- Sort & Search is an interesting general technique,
- Decrease of the worst-case time complexity by increasing the worst-case space complexity,
- Seems to be usable as soon as "objects have to be assigned" (source of the combinatorics).

Conclusions

- Sort & Search is an interesting general technique,
- Decrease of the worst-case time complexity by increasing the worst-case space complexity,
- Seems to be usable as soon as "objects have to be assigned" (source of the combinatorics).

Conclusions

- Sort & Search is an interesting general technique,
- Decrease of the worst-case time complexity by increasing the worst-case space complexity,
- Seems to be usable as soon as "objects have to be assigned" (source of the combinatorics).