Quantifying the hardness of the enumeration of Pareto optima: a theoretical framework with application to scheduling problems

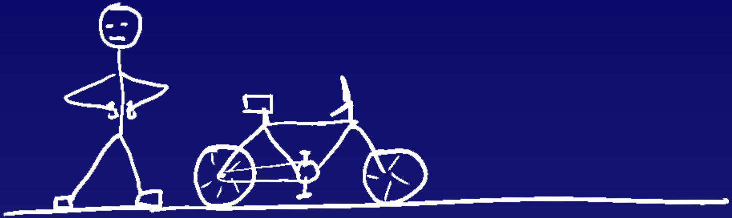
Vincent T'kindt

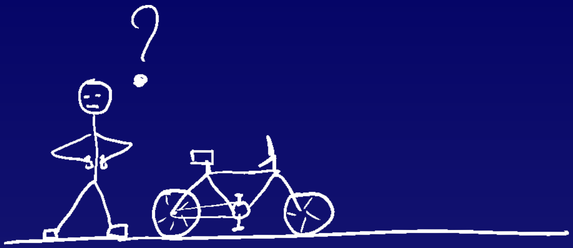
tkindt@univ-tours.fr

University of Tours, Laboratoire d'Informatique Fondamentale et Appliquée (LIFAT EA 6300), ERL CNRS 7002, Tours, France

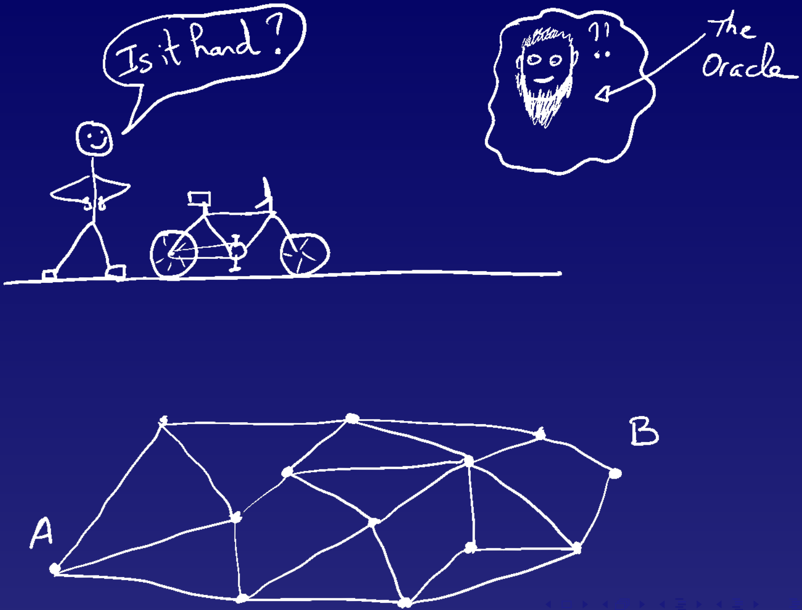
RAMOO workshop, 15th November 2018

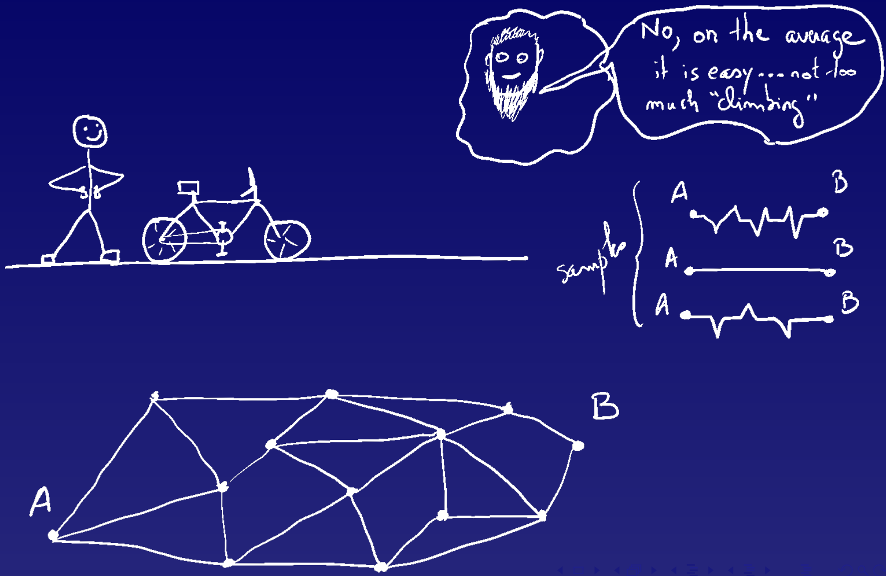
Christophe LENTE (MdC, Lab. d'Informatique Fondamentale et Appliquée, CNRS, University of Tours, Fr),
Lei SHANG (Doctor, Lab. d'Informatique Fondamentale et Appliquée, CNRS, University of Tours, Fr),
Mathieu LIEDLOFF (MdC, Laboratoire d'Informatique Fondamentale d'Orléans, University of Orléans, Fr),
Federico DELLA CROCE (Pr, Politecnico di Torino, It),
Michele GARRAFA (Doctor, Politecnico di Torino, It).



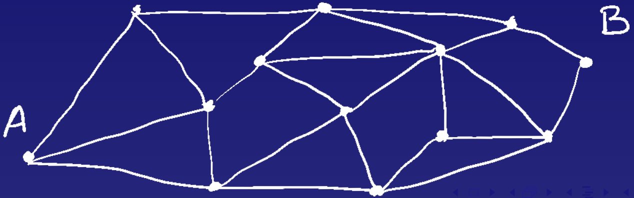


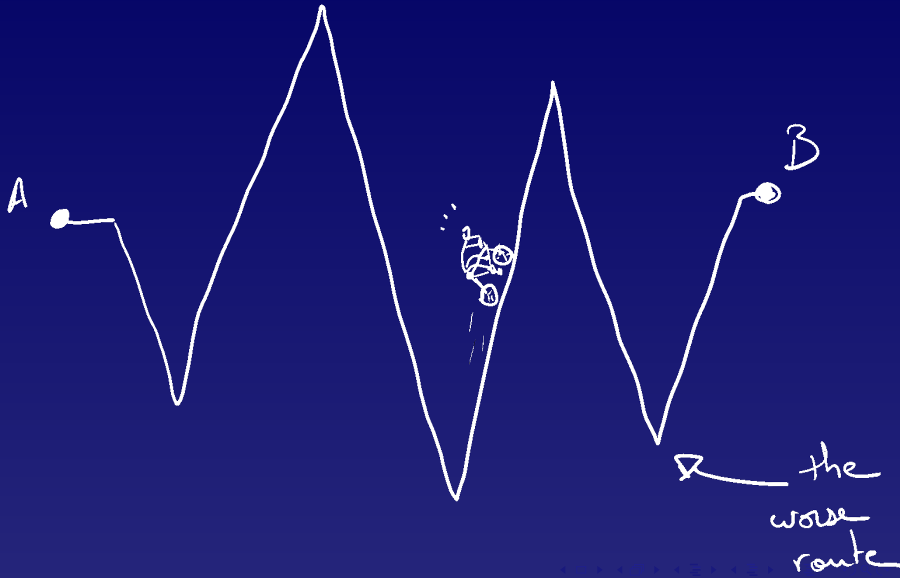

A
B

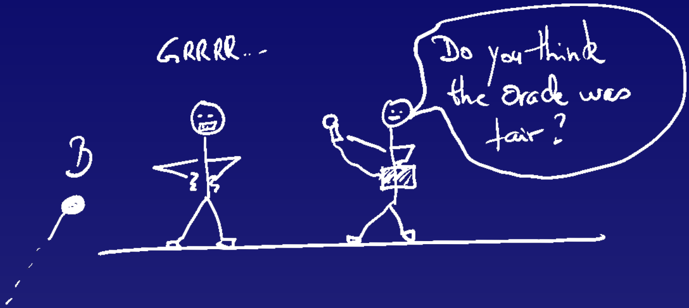

Is it hard ?
The Oracle
?!
A
B


No, on the average
it is easy... not too
much "climbing"
samples
A
B
A
B
A
B
B
A


Great! I go at random!


A
B


A
B
the
worse
route


GRRRR...
Do you think the Oracle was fair?

On a $\mathcal{NP}$-hard problem would it be interesting to have information on its worst-case complexity ?

How can we quantify this complexity ?

$\Rightarrow$ *Exponential-time algorithms* are tools to answer this question.

On a $\mathcal{NP}$-hard problem would it be interesting to have information on its worst-case complexity ?

How can we quantify this complexity ?

$\Rightarrow$ *Exponential-time algorithms* are tools to answer this question.

On a $\mathcal{NP}$-hard problem would it be interesting to have information on its worst-case complexity ?

How can we quantify this complexity ?

$\Rightarrow$ *Exponential-time algorithms* are tools to answer this question.

- What is called an "exponential algorithm"?....
- For a NP-hard problem, an exact algorithm for which *the worst-case (time/space) complexity can be computed*.
- Find "theoretical" algorithms with worst-case time/space upper bounds as low as possible...

The MIS problem has been shown to be solvable $O^*(2^n)$ in 1977, $O^*(1.381^n)$ in 1999, $O^*(1.2201^n)$ in 2009, ...

NB : $O^*(exp(n)) = O(poly(n)exp(n))$

- What is called an “exponential algorithm”?....
- For a NP-hard problem, an exact algorithm for which *the worst-case (time/space) complexity can be computed*.
- Find “theoretical” algorithms with worst-case time/space upper bounds as low as possible...

The MIS problem has been shown to be solvable $O^*(2^n)$ in 1977, $O^*(1.381^n)$ in 1999, $O^*(1.2201^n)$ in 2009, ...

NB : $O^*(exp(n)) = O(poly(n)exp(n))$

- What is called an "exponential algorithm"?....
- For a NP-hard problem, an exact algorithm for which *the worst-case (time/space) complexity can be computed*.
- Find "theoretical" algorithms with worst-case time/space upper bounds as low as possible...

The MIS problem has been shown to be solvable $O^*(2^n)$ in 1977, $O^*(1.381^n)$ in 1999, $O^*(1.2201^n)$ in 2009, ...

NB : $O^*(exp(n)) = O(poly(n)exp(n))$

- Why designing exponential time/space algorithms (ETA) ?
  - The beauty of the game,
  - To provide a quantitative information on the difficulty of a NP-hard problem,
  - Because, in a short future, ETA will start to beat in practice heuristics !

    $O^*(1.2201^n)$ is faster than $O(n^4)$ for $n \leq 90$,

    $O^*(1.1^n)$ is faster than $O(n^4)$ for $n \leq 230$,

- Why designing exponential time/space algorithms (ETA) ?
  - The beauty of the game,
  - To provide a quantitative information on the difficulty of a NP-hard problem,
  - Because, in a short future, ETA will start to beat in practice heuristics !

    $O^*(1.2201^n)$ is faster than $O(n^4)$ for $n \leq 90$,

    $O^*(1.1^n)$ is faster than $O(n^4)$ for $n \leq 230$,

- Why designing exponential time/space algorithms (ETA) ?
  - The beauty of the game,
  - To provide a quantitative information on the difficulty of a NP-hard problem,
  - Because, in a short future, ETA will start to beat in practice heuristics !

    $O^*(1.2201^n)$ is faster than $O(n^4)$ for $n \leq 90$,

    $O^*(1.1^n)$ is faster than $O(n^4)$ for $n \leq 230$,

- Why designing exponential time/space algorithms (ETA) ?
  - The beauty of the game,
  - To provide a quantitative information on the difficulty of a NP-hard problem,
  - Because, in a short future, ETA will start to beat in practice heuristics !

  $O^*(1.2201^n)$ is faster than $O(n^4)$ for $n \leq 90$,

  $O^*(1.1^n)$ is faster than $O(n^4)$ for $n \leq 230$,

- A lot of works on graph or decision problems (70's, 2000-),
  - 3-SAT : $O^*(1.3211^n)$ time (Iwama et al., 2010),
  - Hamiltonian circuit : $O^*(1.657^n)$ time (Bjorklund, 2010),
  - MIS : $O^*(1.2132^n)$ time (Kneis et al, 2009),
  - List coloring : $O^*(2^n)$ time (Bjorklund and Husfeldt, 2006) and (Koivisto, 2006),

- A lot of works on graph or decision problems (70's, 2000-),
  - 3-SAT : $O^*(1.3211^n)$ time (Iwama et al., 2010),
  - Hamiltonian circuit : $O^*(1.657^n)$ time (Bjorklund, 2010),
  - MIS : $O^*(1.2132^n)$ time (Kneis et al, 2009),
  - List coloring : $O^*(2^n)$ time (Bjorklund and Husfeldt, 2006) and (Koivisto, 2006),

- What about scheduling problems (single machine) ?

Problem	brute force	wctc	wcsc	Reference
$1\|dec\|f_{max}$	$O^*(n!)$	$O^*(2^n)$	exp	[1]
$1\|dec\|\sum_i f_i$	$O^*(n!)$	$O^*(2^n)$	exp	[1]
$1\|prec\|\sum_i C_i$	$O^*(n!)$	$O^*((2-\epsilon)^n)$	exp	[2]
$1\|prec\|\sum_i w_i C_i$	$O^*(n!)$	$O^*(2^n)$	exp	[3]
$1\|d_i\|\sum_i w_i U_i$	$O^*(n!)$	$O^*(2^n)$ $O^*(1.4142^n)$	exp exp	[3] [4]
$1\|d_i\|\sum_i T_i$	$O^*(n!)$	$O^*(2^n)$	exp	[3] & [4]
$1\|d_i\|\sum_i w_i T_i$	$O^*(n!)$	$O^*(2^n)$	poly	[5]
$1\|r_i, prec\|\sum_i w_i C_i$	$O^*(n!)$	$O^*(3^n)$	exp	[3] & [4]

[1] F. Fomin, D. Kratsch (2010). Exact Exponential Algorithms, Springer.

[2] M. Cygan, M. Philipczuk, M. Philipczuk, J. Wojtaszczyk (2011). Scheduling partially ordered jobs faster than 2^n, ESA 2011.

[3] G. Woeginger (2003). Exact algorithms for NP-hard problems : A survey, in M. Junger, G. Reinelt, G. Rinaldi (Eds) : Combinatorial Optimization – Eureka I shrink !, Springer, LNCS 2570.

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

[5] M. Garraffa, L . Shang, F. Della Croce, V.T'kindt (2018). An exact exponential branch-and-merge algorithm for the single machine total tardiness problem, Theoretical Computer Science, 745, pp 133-149.

- What about scheduling problems (others) ?

Problem	brute force	wctc	wcsc	Reference
$P\|dec\|f_{max}$	$O^*(m^n n!)$	$O^*(3^n)$	exp	[4]
$P\|dec\|\sum_i f_i$	$O^*(m^n n!)$	$O^*(3^n)$	exp	[4]
$P4\|\|C_{max}$	$O^*(4^n)$	$O^*(2.4142^n)$	exp	[4]
$P3\|\|C_{max}$	$O^*(3^n)$	$O^*(1.7321^n)$	exp	[4]
$P2\|\|C_{max}$	$O^*(2^n)$	$O^*(1.4142^n)$	exp	[4]
$P2\|d_i\|\sum_i w_i U_i$	$O^*(3^n)$	$O^*(1.7321^n)$	exp	[4]
$F2\|\|C^k_{max}$	$O^*(2^n)$	$O^*(1.4142^n)$	exp	[4]
$F3\|\|C_{max}$	$O^*(n!)$	$O^*(3^n)$	exp	[6]
$F3\|\|f_{max}$	$O^*(n!)$	$O^*(5^n)$	exp	[6]
$F3\|\|\sum_i f_i$	$O^*(n!)$	$O^*(5^n)$	exp	[6]
$J2\|\|C^k_{max}$	$O^*(2^n)$	$O^*(1.4142^n)$	exp	[7]

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

[6] L. Shang, C. Lenté, M. Liedloff, V.T'kindt (2016). An exponential dynamic programming algorithm for the 3-machine flowshop scheduling problem to minimize the makespan, Journal of Scheduling, 21(2), pp.227-233.

[7] F. Della Croce, C. Koulamas, V.T'kindt (2016). A constraint generation approach for two-machine shop problems with jobs selection, Eur. J. Oper. Research, submitted.

- We focus on two technics with applications to scheduling :
  - Branch-and-reduce,
  - Sort&Search.
- What happen when multiple objectives are optimized ?

- We focus on two technics with applications to scheduling :
  - Branch-and-reduce,
  - Sort&Search.
- What happen when multiple objectives are optimized ?

Branch-and-... What ? !

- Branch-and-Reduce (BaR) ressembles to known exact algorithms like Branch-and-Bound or Branch-and-Cut...
- A BaR algorithm implements three components :
  - A branching rule,
  - A stopping rule.

Branch-and-... What ? !

- Branch-and-Reduce (BaR) ressembles to known exact algorithms like Branch-and-Bound or Branch-and-Cut...
- A BaR algorithm implements three components :
  - A branching rule,
  - A reduction rule at each node,
  - A stopping rule.

Branch-and-... What ? !

- Branch-and-Reduce (BaR) ressembles to known exact algorithms like Branch-and-Bound or Branch-and-Cut...
- A BaR algorithm implements three components :
  - A branching rule,
  - A reduction rule at each node,
  - A stopping rule.

Branch-and-... What ? !

- Branch-and-Reduce (BaR) ressembles to known exact algorithms like Branch-and-Bound or Branch-and-Cut...
- A BaR algorithm implements three components :
  - A branching rule,
  - A reduction rule at each node,
  - A stopping rule.

Branch-and-... What ? !

- Branch-and-Reduce (BaR) ressembles to known exact algorithms like Branch-and-Bound or Branch-and-Cut...
- A BaR algorithm implements three components :
  - A branching rule,
  - A reduction rule at each node,
  - A stopping rule.

Branch-and-... What ? !

- It is different from known branching algorithms,
  - Lower bounding : not a stopping rule,
    ⇒ We cannot prove that in the worst-case it always prune nodes,
  - Dominance conditions : not reduction rules,
    ⇒ They are only sufficient conditions of optimality.

Branch-and-... What ? !

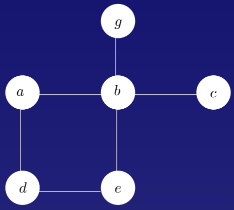
- It is different from known branching algorithms,
  - Lower bounding : not a stopping rule,
    $\Rightarrow$ We cannot prove that in the worst-case it always prune nodes,
  - Dominance conditions : not reduction rules,
    $\Rightarrow$ They are only sufficient conditions of optimality.

Branch-and-... What ? !

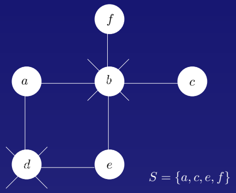
- It is different from known branching algorithms,
  - Lower bounding : not a stopping rule,
    $\Rightarrow$ We cannot prove that in the worst-case it always prune nodes,
  - Dominance conditions : not reduction rules,
    $\Rightarrow$ They are only sufficient conditions of optimality.

Branch-and-Reduce and the MIS

- Consider the *Maximum Independent Set* (MIS) problem :
  Let $G = (V, E)$ be an undirected graph,
  An independent set S is a set of vertices such that no two vertices from S are connected by an edge,
  The MIS problem consists in finding S with a maximum cardinality,



A graph G



A Maximum Independent Set S

Branch-and-Reduce and the MIS

- A first analysis of the problem shows that when $d(v) \leq 2$, $\forall v \in V$, the problem is polynomially solvable,
- This is used as a stopping rule in the Branch-and-Reduce approach (BraRed),
- We now present the BraRed algorithm,

Branch-and-Reduce and the MIS

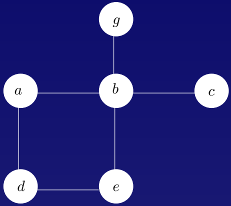
- A first analysis of the problem shows that when $d(v) \leq 2$, $\forall v \in V$, the problem is polynomially solvable,
- This is used as a stopping rule in the Branch-and-Reduce approach (`BraRed`),
- We now present the `BraRed` algorithm,

Branch-and-Reduce and the MIS

- A first analysis of the problem shows that when $d(v) \leq 2$, $\forall v \in V$, the problem is polynomially solvable,
- This is used as a stopping rule in the Branch-and-Reduce approach (`BraRed`),
- We now present the `BraRed` algorithm,

Branch-and-Reduce and the MIS

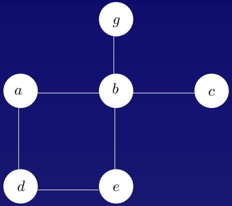
- General case : the maximum degree of vertices is at least 3.



- Let us consider a BraRed algorithm with the following branching rule :
  - Select the vertex v of maximum degree,
  - Create a child node with $v \in S$ and a child node with $v \notin S$.

Branch-and-Reduce and the MIS

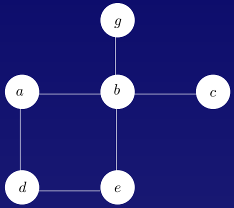
- General case : the maximum degree of vertices is at least 3.



- Let us consider a BraRed algorithm with the following branching rule :
  - Select the vertex v of maximum degree,
  - Create a child node with $v \in S$ and a child node with $v \notin S$.

Branch-and-Reduce and the MIS

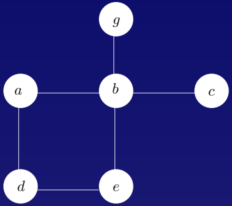
- General case : the maximum degree of vertices is at least 3.



- Let us consider a BraRed algorithm with the following branching rule :
  - Select the vertex v of maximum degree,
  - Create a child node with $v \in S$ and a child node with $v \notin S$.

Branch-and-Reduce and the MIS

- General case : the maximum degree of vertices is at least 3.



- Let us consider a BraRed algorithm with the following branching rule :
  - Select the vertex v of maximum degree,
  - Create a child node with $v \in S$ and a child node with $v \notin S$.

Branch-and-Reduce and the MIS

- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,
- Case 1 : $b \in S$, then a, c, e and f are removed. Vertex $d \in S$ by deduction.
- Case 2 : $b \notin S$, then c and f have degree 0 and are put in S. Vertices a, d, e form a graph of max degree 2... solvable in polynomial time.

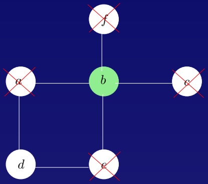
Branch-and-Reduce and the MIS

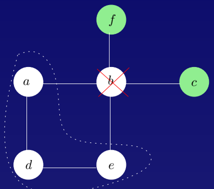
- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,

- Case 1 : $b \in S$, then a, c, e and f are removed. Vertex $d \in S$ by deduction.
- Case 2 : $b \notin S$, then c and f have degree 0 and are put in S. Vertices a, d, e form a graph of max degree 2... solvable in polynomial time.

Branch-and-Reduce and the MIS

- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,

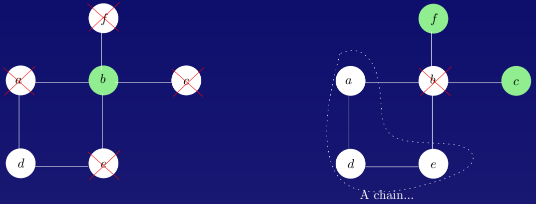




- Case 1 : $b \in S$, then a, c, e and f are removed. Vertex $d \in S$ by deduction.
- Case 2 : $b \notin S$, then c and f have degree 0 and are put in S. Vertices a, d, e form a graph of max degree 2... solvable in polynomial time.

Branch-and-Reduce and the MIS

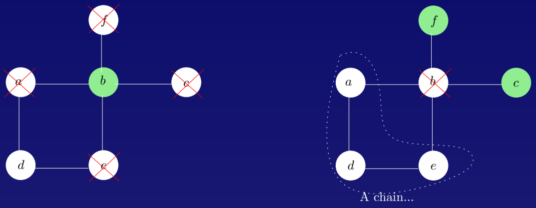
- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,



- Case 1 : $b \in S$, then a, c, e and f are removed. Vertex $d \in S$ by deduction.
- Case 2 : $b \notin S$, then c and f have degree 0 and are put in S. Vertices a, d, e form a graph of max degree 2... solvable in polynomial time.

Branch-and-Reduce and the MIS

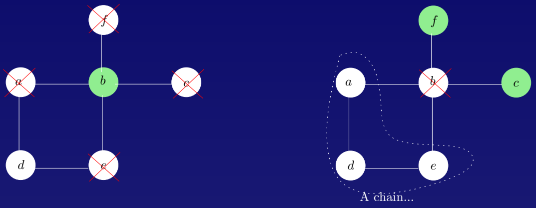
- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,



- In that case 2 nodes have been built.
- Reduction rule : when a decision is taken on a vertex v, decisions are taken for all its neighborhood,
- Stopping rule : for a node, stop branching as far as the maximum degree is 2.

Branch-and-Reduce and the MIS

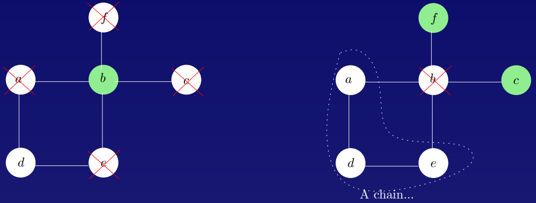
- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,



- In that case 2 nodes have been built.
- Reduction rule : when a decision is taken on a vertex v, decisions are taken for all its neighborhood,
- Stopping rule : for a node, stop branching as far as the maximum degree is 2.

Branch-and-Reduce and the MIS

- General case : the maximum degree of vertices is at least 3.
- Select vertex b of degree 4,



- In that case 2 nodes have been built.
- Reduction rule : when a decision is taken on a vertex v, decisions are taken for all its neighborhood,
- Stopping rule : for a node, stop branching as far as the maximum degree is 2.

Branch-and-Reduce and the MIS

- The `BraRed` algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time at the
- The above processing is applied on any unbranched node in `BraRed`.

Branch-and-Reduce and the MIS

- The `BraRed` algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time on the ...
- The above processing is applied on any unbranched node in `BraRed`.

Branch-and-Reduce and the MIS

- The BraRed algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time at the current node.
- The above processing is applied on any unbranched node in BraRed.

Branch-and-Reduce and the MIS

- The BraRed algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time at the current node.
- The above processing is applied on any unbranched node in BraRed.

Branch-and-Reduce and the MIS

- The `BraRed` algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time at the current node.
- The above processing is applied on any unbranched node in `BraRed`.

Branch-and-Reduce and the MIS

- The `BraRed` algorithm (main iterated loop) :
  - Put all vertices of degree 0 into S,
  - Let v be the vertex with maximum degree :
    - if $d(v) \geq 3$, create two child nodes : one with $v \in S$, another with $v \notin S$. Propagate to its neighborhood.
    - if $d(v) < 3$, solves the problem in polynomial time at the current node.
- The above processing is applied on any unbranched node in `BraRed`.

Branch-and-Reduce and the MIS

- What is the worst-case complexity of `BraRed` ?
- Let us observe the branching rule : $T(n)$ is the time required to solve a problem with n vertices,
- We can state that :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

with v the vertex selected for branching.

- The worst case is obtained when $d(v)$ is minimal, *i.e.* $d(v) = 3$.
- So, in the worst case the time complexity for solving the problem is $T(n) = T(n-4) + T(n-1)$ with $n = |V|$.

Branch-and-Reduce and the MIS

- What is the worst-case complexity of `BraRed` ?
- Let us observe the branching rule : $T(n)$ is the time required to solve a problem with n vertices,
- We can state that :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

with v the vertex selected for branching.
- The worst case is obtained when $d(v)$ is minimal, *i.e.* $d(v) = 3$.
- So, in the worst case the time complexity for solving the problem is $T(n) = T(n-4) + T(n-1)$ with $n = |V|$.

Branch-and-Reduce and the MIS

- What is the worst-case complexity of `BraRed` ?
- Let us observe the branching rule : $T(n)$ is the time required to solve a problem with n vertices,
- We can state that :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

with v the vertex selected for branching.

- The worst case is obtained when $d(v)$ is minimal, *i.e.* $d(v) = 3$.
- So, in the worst case the time complexity for solving the problem is $T(n) = T(n-4) + T(n-1)$ with $n = |V|$.

Branch-and-Reduce and the MIS

- What is the worst-case complexity of `BraRed` ?
- Let us observe the branching rule : $T(n)$ is the time required to solve a problem with n vertices,
- We can state that :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

with v the vertex selected for branching.

- The worst case is obtained when $d(v)$ is minimal, *i.e.* $d(v) = 3$.
- So, in the worst case the time complexity for solving the problem is $T(n) = T(n-4) + T(n-1)$ with $n = |V|$.

Branch-and-Reduce and the MIS

- What is the worst-case complexity of `BraRed` ?
- Let us observe the branching rule : $T(n)$ is the time required to solve a problem with n vertices,
- We can state that :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

  with v the vertex selected for branching.
- The worst case is obtained when $d(v)$ is minimal, *i.e.* $d(v) = 3$.
- So, in the worst case the time complexity for solving the problem is $T(n) = T(n-4) + T(n-1)$ with $n = |V|$.

Branch-and-Reduce and the MIS

- How can we recursively evaluate $T(n) = T(n-4) + T(n-1)$?
- Well, we can use the fact that $T(n) = O^*(x^n)$, with x the largest zero of the function $f(x) = 1 - x^{-4} - x^{-1}$.
- By using a solver like Mathlab (for instance), we obtain $O^*(1.3803^n)$ as the worst-case time complexity for BraRed.
- $O^*(1.3803^n)$ is not bad. Also, BraRed has a polynomial space complexity,

Branch-and-Reduce and the MIS

- How can we recursively evaluate $T(n) = T(n-4) + T(n-1)$?
- Well, we can use the fact that $T(n) = O^*(x^n)$, with x the largest zero of the function $f(x) = 1 - x^{-4} - x^{-1}$.
- By using a solver like Mathlab (for instance), we obtain $O^*(1.3803^n)$ as the worst-case time complexity for BraRed.
- $O^*(1.3803^n)$ is not bad. Also, BraRed has a polynomial space complexity,

Branch-and-Reduce and the MIS

- How can we recursively evaluate $T(n) = T(n-4) + T(n-1)$?
- Well, we can use the fact that $T(n) = O^*(x^n)$, with x the largest zero of the function $f(x) = 1 - x^{-4} - x^{-1}$.
- By using a solver like Mathlab (for instance), we obtain $O^*(1.3803^n)$ as the worst-case time complexity for `BraRed`.
- $O^*(1.3803^n)$ is not bad. Also, `BraRed` has a polynomial space complexity,

Branch-and-Reduce and the MIS

- How can we recursively evaluate $T(n) = T(n-4) + T(n-1)$?
- Well, we can use the fact that $T(n) = O^*(x^n)$, with x the largest zero of the function $f(x) = 1 - x^{-4} - x^{-1}$.
- By using a solver like Mathlab (for instance), we obtain $O^*(1.3803^n)$ as the worst-case time complexity for `BraRed`.
- $O^*(1.3803^n)$ is not bad. Also, `BraRed` has a polynomial space complexity,

Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions S,
- The notion of optimality is here defined by means of Pareto optimality,

Multiobjective optimization

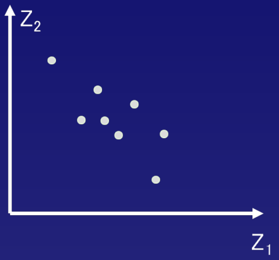
- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

Multiobjective optimization

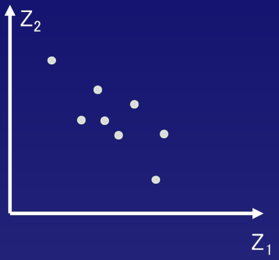
- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

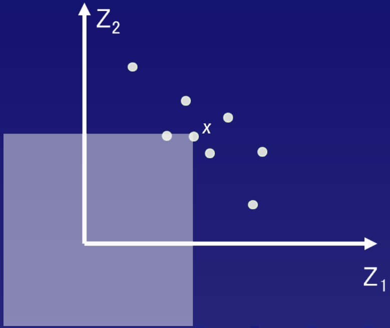
A solution x is a weak Pareto optimum iff there does not exist another solution y such that $Z_i(y) < Z_i(x), \forall i = 1, ..., K$.



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

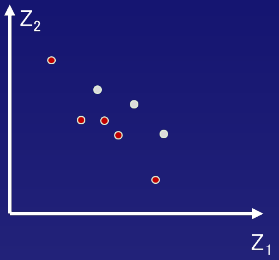
A solution x is a weak Pareto optimum iff there does not exist another solution y such that $Z_i(y) < Z_i(x), \forall i = 1, ..., K$.



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

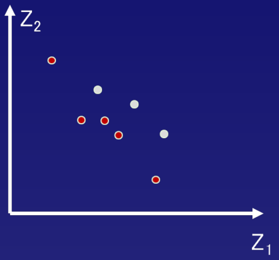
A solution x is a weak Pareto optimum iff there does not exist another solution y such that $Z_i(y) < Z_i(x), \forall i = 1, ..., K$.
$\Rightarrow WE$ is the set of weak Pareto optima (complete set)



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

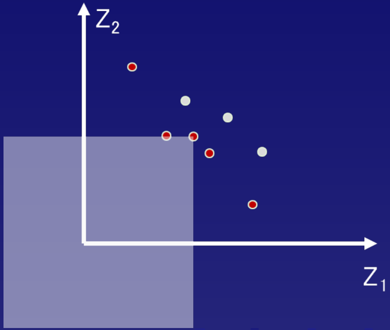
A solution x is a strict Pareto optimum iff there does not exist another solution y such that $Z_i(y) \leq Z_i(x), \forall i = 1, ..., K$, with at least one strict inequality.



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

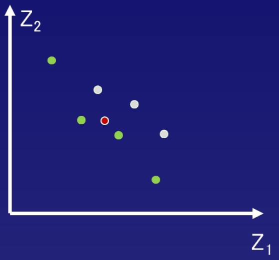
A solution x is a strict Pareto optimum iff there does not exist another solution y such that $Z_i(y) \leq Z_i(x), \forall i = 1, ..., K$, with at least one strict inequality.



Multiobjective optimization

- Let us turn to multiobjective optimization,
- Assume that we have a set of K criteria Z_i to minimise over a set of solutions $\mathcal{S}$,
- The notion of optimality is here defined by means of Pareto optimality,

A solution x is a strict Pareto optimum iff there does not exist another solution y such that $Z_i(y) \leq Z_i(x), \forall i = 1, ..., K$, with at least one strict inequality.
$\Rightarrow E$ is the set of strict Pareto optima (complete set)



Multiobjective optimization

- How to compute a Pareto optimum ?
  1. Convex combination of criteria,
  2. ϵ-constraint approach,
  3. Lexicographic approach,
  4. Parametric approach,
  5. Metric based approaches,
  6. ...

Multiobjective optimization

- How to compute a Pareto optimum ?
  1. Convex combination of criteria,
  2. ϵ-constraint approach,
  3. Lexicographic approach,
  4. Parametric approach,
  5. Metric based approaches,
  6. ...

Multiobjective optimization

- A focus on the ϵ-constraint approach,
- Let us define the following (P_{ϵ}^{k}) problem :

Multiobjective optimization

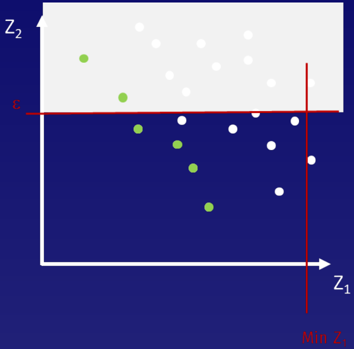
- A focus on the ϵ-constraint approach,
- Let us define the following (P^k_ϵ) problem :

$$\begin{aligned} &\text{Min } Z_k(x) \\ &\quad \text{st} \\ &\qquad x \in \mathcal{S} \\ &\qquad Z_i(x) \leq \epsilon_i^k, \forall i = 1, .., K, \\ &\qquad\qquad i \neq k \end{aligned}$$

Multiobjective optimization

- A focus on the ϵ-constraint approach,
- Let us define the following (P^k_ϵ) problem :

$$\begin{aligned} \text{Min } & Z_k(x) \\ \text{st } & \\ & x \in \mathcal{S} \\ & Z_i(x) \le \epsilon_i^k, \ \forall i = 1, .., K, \\ & \qquad i \neq k \end{aligned}$$



Multiobjective optimization

- A focus on the ϵ-constraint approach,
- Solving, for a fixed k, all (P_ϵ^k) enables to compute a set $E \subseteq WES \subseteq WE$,
- Let us use that approach on a bicriteria scheduling problem.

Multiobjective optimization

- A focus on the ϵ-constraint approach,
- Solving, for a fixed k, all (P_ϵ^k) enables to compute a set $E \subseteq WES \subseteq WE$,
- Let us use that approach on a bicriteria scheduling problem.

Multiobjective optimization

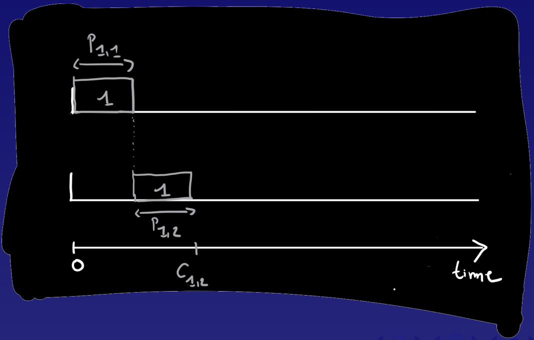
- A focus on the ϵ-constraint approach,
- Solving, for a fixed k, all (P_ϵ^k) enables to compute a set $E \subseteq WES \subseteq WE$,
- Let us use that approach on a bicriteria scheduling problem.

Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).

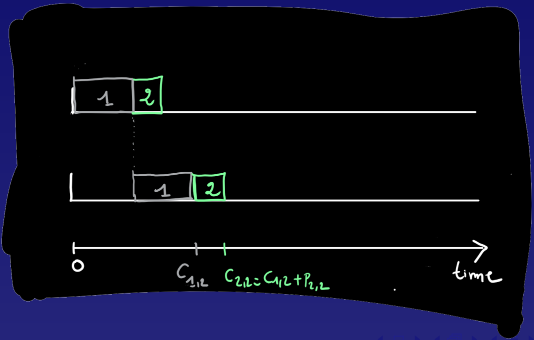
Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



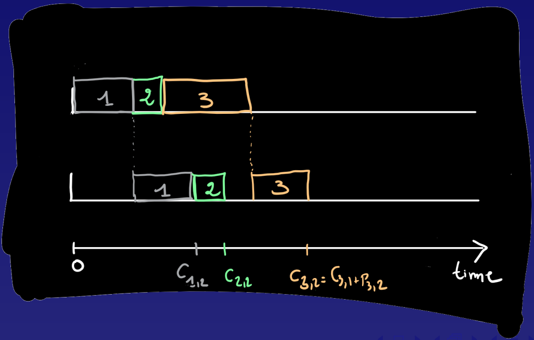
Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



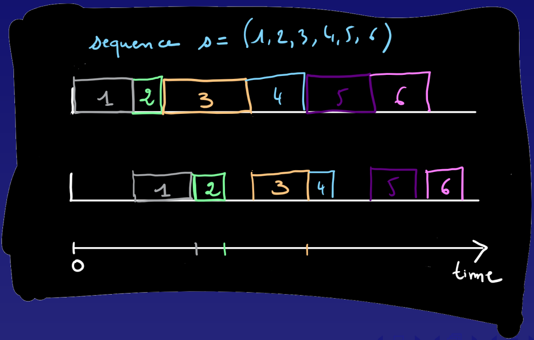
Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



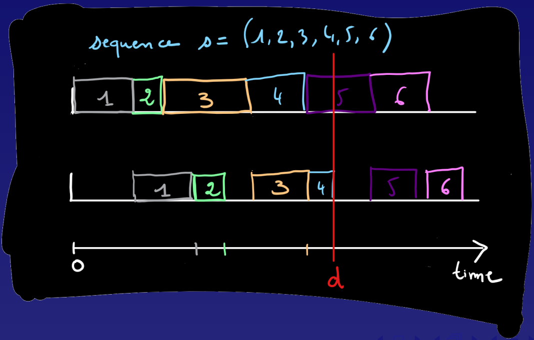
Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



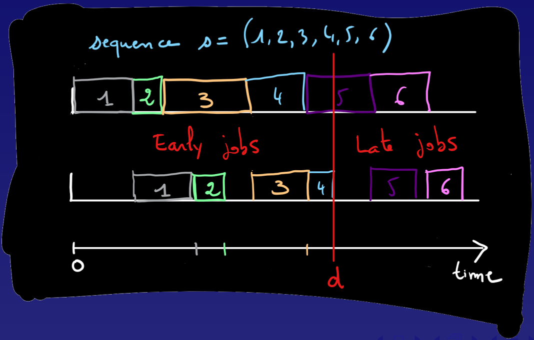
Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



Branch-and-Reduce and multiobjective scheduling

- Let us consider the following scheduling problem :
  - n jobs are to be scheduled on 2 machines,
  - Each job j is defined by $p_{j,1}$, $p_{j,2}$,
  - All jobs share the same common due date d (unknown),
  - Minimize d and $\sum_j U_j$ (number of tardy jobs).



Branch-and-Reduce and multiobjective scheduling

- This problem is referred to as $F2|d_j = d|d, \sum_j U_j$,
- It is $\mathcal{NP}$-hard in the ordinary sense,
- In brief, what we can show :
  - There are exactly $(n+1)$ non dominated criteria vectors,
  - Solving the ϵ-constraint problem :

$$\text{Min } d \text{ st } \sum_j U_j \leq \epsilon$$

    is equivalent to solve :

$$\text{Min } d \text{ st } \sum_j U_j = \epsilon$$

  - If we know the early jobs, d can be computed in poly. time (Johnson's algorithm, 1954).
- So, what's the worst-case time complexity of computing one strict Pareto optimum ?

Branch-and-Reduce and multiobjective scheduling

- This problem is referred to as $F2|d_j = d|d, \sum_j U_j$,
- It is $\mathcal{NP}$-hard in the ordinary sense,
- In brief, what we can show :
  1. There are exactly $(n+1)$ non dominated criteria vectors,
  2. Solving the ϵ-constraint problem :

     $$\text{Min } d \text{ st } \sum_j U_j \leq \epsilon$$

     is equivalent to solve :

     $$\text{Min } d \text{ st } \sum_j U_j = \epsilon.$$

  3. If we know the early jobs, d can be computed in poly. time (Johnson's algorithm, 1954).
- So, what's the worst-case time complexity of computing one strict Pareto optimum ?

Branch-and-Reduce and multiobjective scheduling

- This problem is referred to as $F2|d_j = d|d, \sum_j U_j$,
- It is $\mathcal{NP}$-hard in the ordinary sense,
- In brief, what we can show :
  1. There are exactly $(n+1)$ non dominated criteria vectors,
  2. Solving the ϵ-constraint problem :

     $$\text{Min } d \text{ st } \sum_j U_j \leq \epsilon$$

     is equivalent to solve :

     $$\text{Min } d \text{ st } \sum_j U_j = \epsilon.$$

  3. If we know the early jobs, d can be computed in poly. time (Johnson's algorithm, 1954).
- So, what's the worst-case time complexity of computing one strict Pareto optimum ?

Branch-and-Reduce and multiobjective scheduling

- This problem is referred to as $F2|d_j = d|d, \sum_j U_j$,
- It is $\mathcal{NP}$-hard in the ordinary sense,
- In brief, what we can show :
  1. There are exactly $(n+1)$ non dominated criteria vectors,
  2. Solving the ϵ-constraint problem :

     $$\text{Min } d \text{ st } \sum_j U_j \leq \epsilon$$

     is equivalent to solve :

     $$\text{Min } d \text{ st } \sum_j U_j = \epsilon.$$

  3. If we know the early jobs, d can be computed in poly. time (Johnson's algorithm, 1954).
- So, what's the worst-case time complexity of computing one strict Pareto optimum ?

Branch-and-Reduce and multiobjective scheduling

- This problem is referred to as $F2|d_j = d|d, \sum_j U_j$,
- It is $\mathcal{NP}$-hard in the ordinary sense,
- In brief, what we can show :
  1. There are exactly $(n+1)$ non dominated criteria vectors,
  2. Solving the ϵ-constraint problem :

     $$\text{Min } d \text{ st } \sum_j U_j \leq \epsilon$$

     is equivalent to solve :

     $$\text{Min } d \text{ st } \sum_j U_j = \epsilon.$$

  3. If we know the early jobs, d can be computed in poly. time (Johnson's algorithm, 1954).
- So, what's the worst-case time complexity of computing one strict Pareto optimum ?

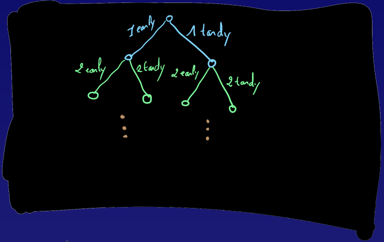
Branch-and-Reduce and multiobjective scheduling

- Let us design the following BraRed algorithm ([7]) :
  - Branching : a job is early or tardy,
  - Reduction : none,
  - Stopping : $\sum_j U_j > \epsilon$.

[7] C. Lenté, M. Liedloff, A. Soukhal, V. T'kindt (2014),*Exponential algorithms for scheduling problems*, HAL, <hal-00944382>.

Branch-and-Reduce and multiobjective scheduling

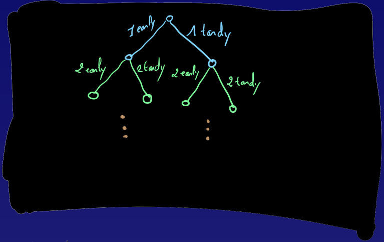
- Let us design the following `BraRed` algorithm ([7]) :
    - Branching : a job is early or tardy,
    - Reduction : none,
    - Stopping : $\sum_j U_j > \epsilon$.



[7] C. Lenté, M. Liedloff, A. Soukhal, V. T'kindt (2014), *Exponential algorithms for scheduling problems*, HAL, <hal-00944382>.

Branch-and-Reduce and multiobjective scheduling

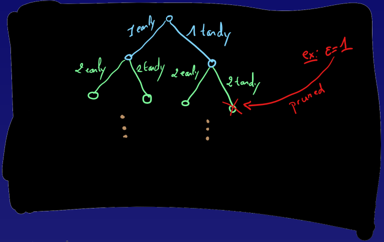
- Let us design the following BraRed algorithm ([7]) :
  - Branching : a job is early or tardy,
  - Reduction : none,
  - Stopping : $\sum_j U_j > \epsilon$.



[7] C. Lenté, M. Liedloff, A. Soukhal, V. T'kindt (2014), *Exponential algorithms for scheduling problems*, HAL, <hal-00944382>.

Branch-and-Reduce and multiobjective scheduling

- Let us design the following BraRed algorithm ([7]) :
  - Branching : a job is early or tardy,
  - Reduction : none,
  - Stopping : $\sum_j U_j > \epsilon$.



[7] C. Lenté, M. Liedloff, A. Soukhal, V. T'kindt (2014), *Exponential algorithms for scheduling problems*, HAL, <hal-00944382>.

Branch-and-Reduce and multiobjective scheduling

- Time complexity analysis :
- Due to the branching scheme, we have :

$$T(n,\epsilon) = T(n-1,\epsilon) + T(n-1,\epsilon-1) = \binom{n}{\epsilon}.$$

- Wlg, assume that $\epsilon = \lambda n$ with $\lambda \in [0;1]$,

Theorem ([7])

BraRed solves the problem with a worst-case time complexity in $O^*([(\frac{1}{\lambda})^{\lambda}(\frac{1}{1-\lambda})^{1-\lambda}]^n)$, *i.e.* $O^*(c(\lambda)^n)$ with $c(\lambda) = (\frac{1}{\lambda})^{\lambda}(\frac{1}{1-\lambda})^{1-\lambda}$, and polynomial space.

Branch-and-Reduce and multiobjective scheduling

- Time complexity analysis :
- Due to the branching scheme, we have :

$$T(n,\epsilon) = T(n-1,\epsilon) + T(n-1,\epsilon-1) = \binom{n}{\epsilon}.$$

- Wlg, assume that $\epsilon = \lambda n$ with $\lambda \in [0;1]$,

Theorem ([7])

BraRed solves the problem with a worst-case time complexity in $O^*([(\frac{1}{\lambda})^\lambda(\frac{1}{1-\lambda})^{1-\lambda}]^n)$, *i.e.* $O^*(c(\lambda)^n)$ with $c(\lambda) = (\frac{1}{\lambda})^\lambda(\frac{1}{1-\lambda})^{1-\lambda}$, and polynomial space.

Branch-and-Reduce and multiobjective scheduling

- Time complexity analysis :
- Due to the branching scheme, we have :

$$T(n,\epsilon) = T(n-1,\epsilon) + T(n-1,\epsilon-1) = \binom{n}{\epsilon}.$$

- Wlg, assume that $\epsilon = \lambda n$ with $\lambda \in [0;1]$,

Theorem ([7])

$BraRed$ solves the problem with a worst-case time complexity in $O^*([(\frac{1}{\lambda})^\lambda(\frac{1}{1-\lambda})^{1-\lambda}]^n)$, *i.e.* $O^*(c(\lambda)^n)$ with $c(\lambda) = (\frac{1}{\lambda})^\lambda(\frac{1}{1-\lambda})^{1-\lambda}$, and polynomial space.

Branch-and-Reduce and multiobjective scheduling

- The result is shown by using Stirling's formulae to approximate $k!$,
- An interesting picture ($\epsilon = \lambda n$),

$\frac{1}{\lambda}$	λ	$c(\lambda)$	Worst-case bound
2	0.50	2	$O^*(2^n)$
3	0.33	1.8898	$O^*(1.8898^n)$
4	0.25	1.7547	$O^*(1.7547^n)$
5	0.20	1.6493	$O^*(1.6493^n)$
6	0.16	1.5691	$O^*(1.5691^n)$
7	0.14	1.5069	$O^*(1.5069^n)$
8	0.12	1.4575	$O^*(1.4575^n)$
9	0.11	1.4174	$O^*(1.4174^n)$
10	0.10	1.3841	$O^*(1.3841^n)$

Branch-and-Reduce and multiobjective scheduling

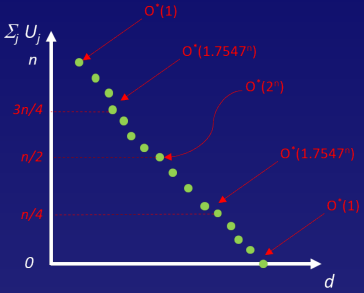
- The result is shown by using Stirling's formulae to approximate $k!$,
- An interesting picture ($\epsilon = \lambda n$),

$\frac{1}{\lambda}$	λ	$c(\lambda)$	Worst-case bound
2	0.50	2	$O^*(2^n)$
3	0.33	1.8898	$O^*(1.8898^n)$
4	0.25	1.7547	$O^*(1.7547^n)$
5	0.20	1.6493	$O^*(1.6493^n)$
6	0.16	1.5691	$O^*(1.5691^n)$
7	0.14	1.5069	$O^*(1.5069^n)$
8	0.12	1.4575	$O^*(1.4575^n)$
9	0.11	1.4174	$O^*(1.4174^n)$
10	0.10	1.3841	$O^*(1.3841^n)$

Branch-and-Reduce and multiobjective scheduling

- The result is shown by using Stirling's formulae to approximate $k!$,
- An interesting picture ($\epsilon = \lambda n$),

$\frac{1}{\lambda}$	λ	$c(\lambda)$	Worst-case bound
2	0.50	2	$O^*(2^n)$
3	0.33	1.8898	$O^*(1.8898^n)$
4	0.25	1.7547	$O^*(1.7547^n)$
5	0.20	1.6493	$O^*(1.6493^n)$
6	0.16	1.5691	$O^*(1.5691^n)$
7	0.14	1.5069	$O^*(1.5069^n)$
8	0.12	1.4575	$O^*(1.4575^n)$
9	0.11	1.4174	$O^*(1.4174^n)$
10	0.10	1.3841	$O^*(1.3841^n)$



Branch-and-Reduce : to conclude

- We have upper bounds on the worst-case time complexity for the flowshop problem,
- A brute force enumeration approach Enum solves the problem in $O^*(2^n)$ time and polynomial space,
- A Sort & Search approach solves the problem in $O^*(1.4143^n)$ time... and space.

Branch-and-Reduce : to conclude

- We have upper bounds on the worst-case time complexity for the flowshop problem,
- A brute force enumeration approach `Enum` solves the problem in $O^*(2^n)$ time and polynomial space,
- A Sort & Search approach solves the problem in $O^*(1.4143^n)$ time... and space.

Branch-and-Reduce : to conclude

- We have upper bounds on the worst-case time complexity for the flowshop problem,
- A brute force enumeration approach `Enum` solves the problem in $O^*(2^n)$ time and polynomial space,
- A Sort & Search approach solves the problem in $O^*(1.4143^n)$ time... and space.

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 ([8]) to solve the knapsack problem (SCP),
- It has been extended by Lenté et al. ([4]) to solve *Multiple Constraint Problems* (MCP),

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

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

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 ([8]) to solve the knapsack problem (SCP),
- It has been extended by Lenté et al. ([4]) to solve *Multiple Constraint Problems* (MCP),

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

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

Sort & Search : the principles

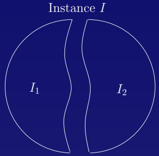
- It is an old technique which consists in **sorting** “data” to make the **search** for an optimal solution more efficient,
- It has been proposed by Horowitz and Sahni ([8]) to solve the knapsack problem (SCP),
- It has been extended by Lenté et al. ([4]) to solve *Multiple Constraint Problems* (MCP),

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

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

Sort & Search : the principles

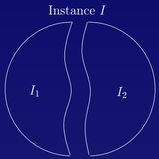
- The idea is the following : separate the instance into **2** sub-instances,



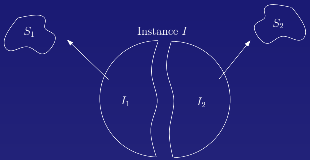
- Then, enumerate all partial solutions from I_1 and all partial solutions from I_2,

Sort & Search : the principles

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

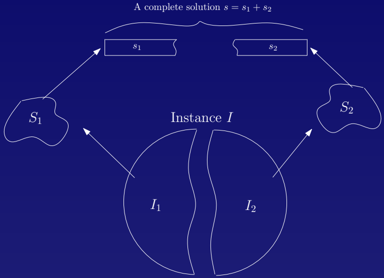


- Then, enumerate all partial solutions from I_1 and all partial solutions from I_2,



Sort & Search : the principles

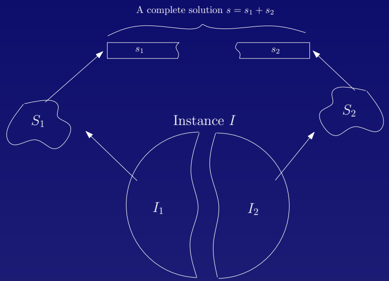
- By recombination of partial solutions, find the optimal solution of the initial problem



- 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 and multiobjective optimization

- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- This approach has been extended by Shang and T'kindt ([9]) to solve *Multiobjective MCP* (MMCP),
- So, what is the look of a MMCP ?

[9] L. Shang and V. T'kindt (2018). A *Sort & Search* method for multicriteria optimization problems with applications to scheduling theory, submitted, 16 pages.

Sort & Search and multiobjective optimization

- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- This approach has been extended by Shang and T'kindt ([9]) to solve *Multiobjective MCP* (MMCP),
- So, what is the look of a MMCP ?

[9] L. Shang and V. T'kindt (2018). A *Sort & Search* method for multicriteria optimization problems with applications to scheduling theory, submitted, 16 pages.

Sort & Search and multiobjective optimization

- The idea : cut the cake into two equal-size pieces and just pay for one (but take both !),
- This approach has been extended by Shang and T'kindt ([9]) to solve *Multiobjective MCP* (MMCP),
- So, what is the look of a MMCP ?

[9] L. Shang and V. T'kindt (2018). A *Sort & Search* method for multicriteria optimization problems with applications to scheduling theory, submitted, 16 pages.

Sort & Search and multiobjective optimization

- The (MMCP) can be defined as follows :

$$
\begin{aligned}
&\text{Minimize } f_1(\boldsymbol{a_j}, b_k^1)\\
&\ldots\\
&\text{Minimize } f_K(\boldsymbol{a_j}, b_k^K)\\
&\text{s.t.}\\
&\quad g_\ell(\boldsymbol{a_j}, b_k^{K+\ell}) \geq 0, \quad (1 \leq \ell \leq d_B)\\
&\quad \boldsymbol{a}_j \in A,\ \boldsymbol{b}_k \in B.
\end{aligned}
$$

with A a table of n_A vectors of dimension d_A, B a table of n_B vectors of dimension $(d_B + K)$, f_h $(1 \leq h \leq K)$ and g_ℓ $(1 \leq \ell \leq d_B)$, $(d_B + K)$ functions from $\mathbb{R}^{d_A+1}$ to $\mathbb{R}$ which are non-decreasing with respect to their last variable.

Sort & Search and multiobjective optimization

Theorem ([9])

If a multiobjective optimization problem can be reformulated as a (MMCP) then there exists a *Sort & Search* algorithm to solve it and which requires $O(|E| \cdot (K n_B \log_2^{d_B+2}(n_B) + K 2^K))$ time and $O(n_B \log_2^{d_B-1}(n_B) + |E|)$ space.

- Example : the $P2|d_i|C_{\max}, L_{\max}$ problem,
  - $K = 2$, $d_B = 3$, $n_A = n_B = n/2$,
  - Worst-case time complexity is $O^*(|E| \cdot 1.4143^n)$ and the space complexity is $O^*(1.4143^n + |E|)$,
  - As $|E| \leq 2^n$, the time complexity is $O^*(2.83^n)$ and the space complexity is $O^*(2^n)$.

Sort & Search and multiobjective optimization

Theorem ([9])

If a multiobjective optimization problem can be reformulated as a (MMCP) then there exists a *Sort & Search* algorithm to solve it and which requires $O(|E| \cdot (Kn_B \log_2^{d_B+2}(n_B) + K2^K))$ time and $O(n_B \log_2^{d_B-1}(n_B) + |E|)$ space.

- Example : the $P2|d_i|C_{\max}, L_{\max}$ problem,
  - $K = 2$, $d_B = 3$, $n_A = n_B = 2^{\frac{n}{2}}$,
  - Worst-case time complexity is $O^*(|E| \cdot 1.4143^n)$ and the space complexity is $O^*(1.4143^n + |E|)$,
  - As $|E| \leq 2^n$, the time complexity is $O^*(2.83^n)$ and the space complexity is $O^*(2^n)$.

Sort & Search and multiobjective optimization

Theorem ([9])

If a multiobjective optimization problem can be reformulated as a (MMCP) then there exists a *Sort & Search* algorithm to solve it and which requires $O(|E| \cdot (Kn_B \log_2^{d_B+2}(n_B) + K2^K))$ time and $O(n_B \log_2^{d_B-1}(n_B) + |E|)$ space.

- Example : the $P2|d_i|C_{\max}, L_{\max}$ problem,
  - $K = 2,\ d_B = 3,\ n_A = n_B = 2^{\frac{n}{2}}$,
  - Worst-case time complexity is $O^*(|E| \cdot 1.4143^n)$ and the space complexity is $O^*(1.4143^n + |E|)$,
  - As $|E| \leq 2^n$, the time complexity is $O^*(2.83^n)$ and the space complexity is $O^*(2^n)$.

Sort & Search and multiobjective optimization

Theorem ([9])

If a multiobjective optimization problem can be reformulated as a (MMCP) then there exists a *Sort & Search* algorithm to solve it and which requires $O(|E| \cdot (Kn_B \log_2^{d_B+2}(n_B) + K2^K))$ time and $O(n_B \log_2^{d_B-1}(n_B) + |E|)$ space.

- Example : the $P2|d_i|C_{\max}, L_{\max}$ problem,
  - $K = 2,\ d_B = 3,\ n_A = n_B = 2^{\frac{n}{2}}$,
  - Worst-case time complexity is $O^*(|E| \cdot 1.4143^n)$ and the space complexity is $O^*(1.4143^n + |E|)$,
  - As $|E| \leq 2^n$, the time complexity is $O^*(2.83^n)$ and the space complexity is $O^*(2^n)$.

Sort & Search and multiobjective optimization

Theorem ([9])

If a multiobjective optimization problem can be reformulated as a (MMCP) then there exists a *Sort & Search* algorithm to solve it and which requires $O(|E| \cdot (Kn_B \log_2^{d_B+2}(n_B) + K2^K))$ time and $O(n_B \log_2^{d_B-1}(n_B) + |E|)$ space.

- Example : the $P2|d_i|C_{\max}, L_{\max}$ problem,
  - $K = 2,\ d_B = 3,\ n_A = n_B = 2^{\frac{n}{2}}$,
  - Worst-case time complexity is $O^*(|E| \cdot 1.4143^n)$ and the space complexity is $O^*(1.4143^n + |E|)$,
  - As $|E| \leq 2^n$, the time complexity is $O^*(2.83^n)$ and the space complexity is $O^*(2^n)$.

Sort & Search : to conclude

- *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 : to conclude

- *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 : to conclude

- *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 : to conclude

- *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.

Conclusions

- Exponential Time Algorithms provide us with worst-case information,
- Ok, finding ETA with reduced worst-case complexities is a challenging (theoretical) issue,
- But, this provides insights on "quantifying the hardness of problems"... do all $\mathcal{NP}$-hard problems have the same complexity in the worst-case ?

Conclusions

- Exponential Time Algorithms provide us with worst-case information,
- Ok, finding ETA with reduced worst-case complexities is a challenging (theoretical) issue,
- But, this provides insights on "quantifying the hardness of problems"... do all $\mathcal{NP}$-hard problems have the same complexity in the worst-case ?

Conclusions

- Exponential Time Algorithms provide us with worst-case information,
- Ok, finding ETA with reduced worst-case complexities is a challenging (theoretical) issue,
- But, this provides insights on "quantifying the hardness of problems"... do all $\mathcal{NP}$-hard problems have the same complexity in the worst-case ?

Conclusions

- What can you retrieve from such a presentation ?
  - Multiobjective optimization is a totally unexplored area as far as ETA algorithms are concerned...
  - ... despite the importance of the question :

    Do all Pareto optima require the same "complexity" to be computed ?
  - On a practical side : study the worst-case behavior of your exact algorithms to improve when they do not work ("branching" issues),

Conclusions

- What can you retrieve from such a presentation ?
  - Multiobjective optimization is a totally unexplored area as far as ETA algorithms are concerned...
  - ... despite the importance of the question :

    Do all Pareto optima require the same "complexity" to be computed ?

  - On a practical side : study the worst-case behavior of your exact algorithms to improve when they do not work ("branching" issues),

Conclusions

- What can you retrieve from such a presentation ?
  - Multiobjective optimization is a totally unexplored area as far as ETA algorithms are concerned...
  - ... despite the importance of the question :

    Do all Pareto optima require the same "complexity" to be computed ?
  - On a practical side : study the worst-case behavior of your exact algorithms to improve when they do not work ("branching" issues),

Conclusions

- What can you retrieve from such a presentation ?
  - Multiobjective optimization is a totally unexplored area as far as ETA algorithms are concerned...
  - ... despite the importance of the question :

    Do all Pareto optima require the same "complexity" to be computed ?
  - On a practical side : study the worst-case behavior of your exact algorithms to improve when they do not work ("branching" issues),