## All meals for a dollar and other vertex enumeration problems

David Avis

Kyoto University and McGill University http://cgm.cs.mcgill.ca/~avis

October 18, 2018

Outline Vertex Enumeration Reverse Search Parallel Reverse

Vertex Enumeration

Reverse Search

Parallel Reverse Search

Reverse Search Parallel Reverse Search

Vertex Enumeration

Outline

Outline of talk

#### Diet problem

• Situation: You need to choose some food in the supermarket to feed yourself properly for just \$1 per day.

Parallel Reverse Search			
Keverse Search			
Vertex Enumeration			
Outline			

#### Diet problem

- Situation: You need to choose some food in the supermarket to feed yourself properly for just \$1 per day.
  - Decison variables: How much of each product you will buy.



Parallel Reverse Search	
Reverse Search	
Vertex Enumeration	
Outline	

#### Diet problem

- Situation: You need to choose some food in the supermarket to feed yourself properly for just \$1 per day.
- Decison variables: How much of each product you will buy.
- Constraints: There are minimum daily requirements for calories, vitamins, calcium, etc. There is a maximum amount of each food you can eat.



Outline	Vertex Enumeration	Reverse Search	Parallel Reverse Search	

#### Diet problem

- Situation: You need to choose some food in the supermarket to feed yourself properly for just \$1 per day.
- Decison variables: How much of each product you will buy.
- Constraints: There are minimum daily requirements for calories, vitamins, calcium, etc. There is a maximum amount of each food you can eat.
- Objective Eat for less than \$1.

#### Sample data

Max	Serv.	4	3	2	∞	2	2	
Price	÷	က	24	13	6	20	19	
Calcium	(mg)	2	12	54	285	22	80	800
Protein	(g)	4	32		∞	4	14	22
Energy	(kcal)	110	202	160	160	420	260	2000
Serv.	Size	28g	100g	2 large	237ml	170g	260g	
Food		Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork w. beans	Min. Daily Amt.
		×	×27	ž	× <sub>4</sub>	×	% %	

The decision variables are  $x_1, x_2, ..., x_6$ . Fractional servings are allowed. From *Linear Programming*, Vasek Chvátal, 1983

Parallel Reverse Search

# Linear programming formulation for diet problem

	Food	Serv.	Energy	Protein	Calcium	Price	Max
		Size	(kcal)	(g)	(mg)	÷	Serv.
Oatmea	neal	28g	110	4	2	3	4
Chic	Chicken	100g	202	32	12	24	က
யூ	Eggs	2 large	160	13	54	13	7
Milk	≝	237ml	160	∞	285	6	∞
Cher	ry Pie	170g	420	4	22	20	7
Pork w	Pork w. beans	260g	260	14	8	19	7
Min. D	Daily Amt.		2000	22	800		

$$\begin{aligned} & \textit{min } z &= 3x_1 \, + \, 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \\ \text{s.t.} & 110x_1 \, + \, 205x_2 \, + \, 160x_3 + 160x_4 + 420x_5 + 260x_6 \, \geq \, 2000 \\ & 4x_1 \, + \, 32x_2 \, + \, 13x_3 + 8x_4 + 4x_5 + 14x_6 \, \geq \, 55 \\ & 2x_1 \, + \, 12x_2 \, + \, 54x_3 + 285x_4 + 22x_5 + 80x_6 \, \geq \, 800 \\ & 0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \\ & 0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2 \end{aligned}$$

## Linear programming solution

		_						
Max	Serv.	4	3	7	∞	7	7	
Price	÷	3	24	13	6	20	19	
Calcium	(mg)	2	12	54	285	22	80	800
Protein	(g)	4	32	13	∞	4	14	22
Energy	(kcal)	110	202	160	160	420	260	2000
Serv.	Size	28g	100g	2 large	237ml	170g	260g	
Food		Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork w. beans	Min. Daily Amt.
		×1	×2	×	¥	×5	××	

Reverse Search

באפואם

## Linear programming solution

_		_						$\overline{}$
Max	Serv.	4	က	2	∞	2	7	
Price	÷	3	24	13	6	20	19	
Calcium	(mg)	2	12	54	285	22	8	800
Protein	(g)	4	32	13	∞	4	14	22
Energy	(kcal)	110	202	160	160	420	260	2000
Serv.	Size	28g	100g	2 large	237ml	170g	260g	
Food		Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork w. beans	Min. Daily Amt.
		¥	×	z.	¥	× <sub>5</sub>	9X	

•  $x_1 = 4(\text{oatmeal}) \ x_4 = 4.5(\text{milk}) \ x_5 = 2(\text{pie}) \ \text{cost} = 92.5 \ \text{¢}$ 

## Linear programming solution

×	<u>.</u>	4	~	~	<u> </u>	~	7	
Max	Serv.							
Price	<del>U</del>	က	24	13	6	20	19	
Calcium	(mg)	2	12	54	285	22	80	800
Protein	(g)	4	32	13	∞	4	14	22
Energy	(kcal)	110	202	160	160	420	260	2000
Serv.	Size	28g	100g	2 large	237ml	170g	260g	
Food		Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork w. beans	Min. Daily Amt.
		¥	×	z.	¥	X <sub>5</sub>	9X	

- $x_1 = 4(\text{oatmeal}) \ x_4 = 4.5(\text{milk}) \ x_5 = 2(\text{pie}) \ \text{cost} = 92.5 \ \text{¢}$ 
  - Where are the chicken, eggs and pork?

Linear programming solution

_		_						_
Max	Serv.	4	n	2	∞	2	2	
Price	÷	3	24	13	6	20	19	
Calcium	(mg)	2	12	54	285	22	80	800
Protein	(g)	7	32	13	∞	4	14	22
Energy	(kcal)	110	202	160	160	420	260	2000
Serv.	Size	28g	100g	2 large	237ml	170g	260g	
Food		Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork w. beans	Min. Daily Amt.
		×	× <sup>2</sup>	£	¥	× <sub>5</sub>	% %	

- $x_1=4(\text{oatmeal})\ x_4=4.5(\text{milk})\ x_5=2(\text{pie})\ \text{cost}=92.5\ \varphi$  Where are the chicken, eggs and pork?
- Do I have to eat the same food every day?

Parallel Keverse Search		
Keverse Search		Problems with the solition
Vertex Enumeration	-	Problems w
Outline		



Parallel Reverse Search		
Keverse Search		the sollition
Vertex Enumeration		 Problems with the solution
Outline		

• Many desirable items were not included in the optimum solution



se pearon		
Larallel Dever		2
Leverse Search		Problems with the solution
vertex Enumeration		Problems
Outillie		

- Many desirable items were not included in the optimum solution
  - We obtained a unique optimum solution, but ...



the solution
the solution
Problems with the solution

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
  - ... people (and managers) like to make choices!



raiallei iveveise sealell		
Iveverse dealer		the solution
vertex Elimineration		 Problems with the solution
Outillie		

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
  - Ask the right question!



Parallel Reverse Search	
Reverse Search	Problems with the solution
Vertex Enumeration	Problems wi
Outiline	

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
- Ask the right question!
- $\bullet$  What are all the meals I can eat for at most \$1?

Replace the objective function by an inequality:

$$3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \le 100$$

$$110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \ge 2000$$

$$4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \ge 55$$

$$2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \ge 800$$

$$0 \le x_1 \le 4, \quad 0 \le x_2 \le 3, \quad 0 \le x_3 \le 2,$$

$$0 \le x_4 \le 8, \quad 0 \le x_5 \le 2, \quad 0 \le x_6 \le 2$$

Replace the objective function by an inequality:

$$3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \le 100$$

$$110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \ge 2000$$

$$4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \ge 55$$

$$2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \ge 800$$

$$0 \le x_1 \le 4, \quad 0 \le x_2 \le 3, \quad 0 \le x_3 \le 2,$$

$$0 \le x_4 \le 8, \quad 0 \le x_5 \le 2, \quad 0 \le x_6 \le 2$$

 $\bullet$  Any solution to these inequalities is a meal for under \$1

Replace the objective function by an inequality:

$$3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \le 100$$

$$110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \ge 2000$$

$$4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \ge 55$$

$$2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \ge 800$$

$$0 \le x_1 \le 4, \quad 0 \le x_2 \le 3, \quad 0 \le x_3 \le 2,$$

$$0 \le x_4 \le 8, \quad 0 \le x_5 \le 2, \quad 0 \le x_6 \le 2$$

- $\bullet$  Any solution to these inequalities is a meal for under \$1
- But this is just a restatement of the problem ......

Replace the objective function by an inequality:

$$3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \le 100$$

$$110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \ge 2000$$

$$4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \ge 55$$

$$2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \ge 800$$

$$0 \le x_1 \le 4, \quad 0 \le x_2 \le 3, \quad 0 \le x_3 \le 2,$$

$$0 \le x_4 \le 8, \quad 0 \le x_5 \le 2, \quad 0 \le x_6 \le 2$$

- $\bullet$  Any solution to these inequalities is a meal for under \$1
- But this is just a restatement of the problem ......
- ... how do I find these solutions?

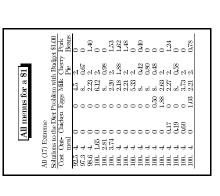
### A more useful solution

### A more useful solution

		00	l,	S	Ī																	1
		get \$1.0	y Pork	Beans	0	0	1.40	0	0	1.53	1,62	1.48	0	9	0	0	0	1.24	0	0	0.78	
\$1		th Bud	Cherry	Þ.	7	0.67		4	0.98					0.42					0.58	c1	2	
for a		Jen wi	s Milk		4.5	œ	2.23	6.12	œ	2.20	2.18	2.21	5.33	ø	œ	80		2.27	ø.	3.73		
All menus for		et Prol	en Eggs		0	0	0	0	0	0	0	0	0	0	0	0.50	88	0	0	0	1.03	
All m	All (17) Extreme	Solutions to the Diet Problem with Budget \$1.00	Chicken		0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0.19	09.0	0	
	17) Ex	ions to	Oat-	mea	4	4	4				4					4				4	4	
	All (	Solut	Cost		92.5	97.3	98.6	100	100	100	100	100	100	100	100	100	100	100	100	9	100	

Outline Vertex Enumeration Reverse Search Parallel Reverse Search

### A more useful solution



Taking convex combinations of rows gives new meals

Vertex Enumeration Reverse Search Farallel Neverse

### A more useful solution

		\$1.00	Pork	Beans	0	0	1.40	0	0	1.53	1.62	1.48	0	0.40	0	0	0	1.24	0	0	0.78	
\$1		solutions to the Diet Problem with Budget \$1.00	YII.	Pe.	7	0.67	2	2	0.98	2	88	2	2	0.42	0.80	0.48	7	7	0.58	7	7	
		em with			4.5	œ										œ	2.63				2.21	
enus 1		t Probl	hiden Eggs Milk		0	0	0	0	0	0	0	0	0	0	0	0.50	88	0	0	0	1.03	
All menus for a	treme	the Di	$\sim$		0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0.19	090	0	
	All (17) Extreme	ions to	Oat-	шеа	Ŧ	4	4		2.81		7	4	4						7		4	
	All (	Solut	Cost		92.5	97.3	98.6	100	100	100	100	100	100	100	100	100	100	100	100	100	100	

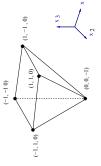
- Taking convex combinations of rows gives new meals
- Eg. Taking half each of the last two rows gives a \$1 meal with all foods

Vertex Enumeration

Parallel Reverse Search

10





H-representation:  $| -x_1 - x_3 \ge 0$   $| -x_2 + x_3 \ge 0$   $| +x_1 - x_2 + x_3 \ge 0$   $| +x_1 - x_3 \ge 0$   $| +x_2 - x_3 \ge 0$ 

 $-x_3 \ge 0$ 

V-representation:  $v_1=(-1,1,0), \quad v_2=(-1,-1,0), \quad v_3=(1,-1,0),$   $v_4=(1,1,0), \quad v_5=(0,0,-1)$ 

raiaiei Neverse Searcii	olyhedron
Neverse Sealer	of a bounded po
Vertex Endineration	Two representations of a bounded polyhedron
Outille	Ϋ́

Two representations of a bounded polyhedron	
Two representations of a bounded bo	lyhedron
Two representations of a	bounded pol
Two	epresentations of a
	Two

• H-representation (Half-spaces):  $\{x \in R^n : Ax \le b\}$ 

• H-representation (Half-spaces):  $\{x \in R^n : Ax \le b\}$ • V-representation (Vertices):  $v_1, v_2, ..., v_N$  are the vertices of P

$$x=\sum_{i=1}^{N}\lambda_iv_i$$
 where  $\sum_{i=1}^{N}\lambda_i=1, \quad \lambda_i\geq 0, \ i=1,2,...,N$ 

- H-representation (Half-spaces):  $\{x \in R^n: Ax \le b\}$  V-representation (Vertices):  $v_1, v_2, ..., v_N$  are the vertices of P

$$x = \sum_{i=1}^N \lambda_i \nu_i$$
 where  $\sum_{j=1}^N \lambda_i = 1, \quad \lambda_i \geq 0, \ i=1,2,...,N$ 

• Vertex enumeration: H-representation  $\Rightarrow$  V-representation

- H-representation (Half-spaces):  $\{x \in R^n : Ax \le b\}$
- V-representation (Vertices):  $v_1, v_2, ..., v_N$  are the vertices of P

$$x = \sum_{i=1}^N \lambda_i \nu_i$$
 where  $\sum_{i=1}^N \lambda_i = 1, \quad \lambda_i \geq 0, \ i=1,2,...,N$ 

- Vertex enumeration: H-representation  $\Rightarrow$  V-representation
- Convex hull problem: V-representation  $\Rightarrow$  H-representation

- H-representation (Half-spaces):  $\{x \in R^n : Ax \le b\}$
- V-representation (Vertices):  $v_1, v_2, ..., v_N$  are the vertices of P

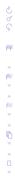
$$x = \sum_{i=1}^N \lambda_i v_i$$
 where  $\sum_{i=1}^N \lambda_i = 1, \quad \lambda_i \geq 0, \ i=1,2,...,N$ 

- Vertex enumeration: H-representation  $\Rightarrow$  V-representation
- $\bullet$  Convex hull problem: V-representation  $\Rightarrow$  H-representation
- Solution methods: double description(cdd) and reverse search(Irs)

Parallel Reverse Search	
Reverse Search	ex enumeration?
Vertex Enumeration	Who uses vertex enumeration?
Outline	

ו מומופו ויפעפואם ספמיכו	
	Who uses vertex enumeration?
Vertex Francisco	Who uses ver

• Wide variety of users: scientists, engineers, economists, operations researchers ...



Vertex Enumeration

- Wide variety of users: scientists, engineers, economists, operations researchers ...
  - ...who are not experts in polyhedral computation ...



Parallel Keverse Search		
Keverse Search	C	Who uses vertex enumeration?
Vertex Enumeration	1///	vvno uses verte
Outline		

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers



Parallel Reverse Search		
Keverse Search	-	x enumeration?
Vertex Enumeration	17.4.1	Who uses vertex enumeration
Outline		

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
- $\bullet$  Software should be easy to install, run on standard work stations and  $\dots$



Parallel Reverse Search	
Reverse Search	Who uses vertex enumeration?
Vertex Enumeration	Who uses ver
Outline	

 Wide variety of users: scientists, engineers, economists, operations researchers ...

...who are not experts in polyhedral computation ...

... and not software engineers

 $\bullet$  Software should be easy to install, run on standard work stations and  $\dots$ 

• ... should run faster on better hardware!

Parallel Keverse Search	
Keverse Search	x enumeration?
Vertex Enumeration	Who uses vertex enumeration?
Outline	

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
- Software should be easy to install, run on standard work stations and  $\ldots$
- ... should run faster on better hardware!
- Goal: parallelize *Irs* for multicore workstations using existing code

Reverse Search Vertex Enumeration

Case study: MIT problem

#### PHYSICAL REVIEW B CONDENSED MATTER

THIRD SERIES, VOLUME 49, NUMBER 1

1 JANUARY 1994-I

# Ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor interactions

G. Ceder and G. D. Garbuisky
Department of Materials Science and Engineering, Massachusetts 11199

Department of Materials Science and Engineering, Massachusetts 11199

D. Avis School of Computer Science, McGill University, Montreal, Quebec, Canada H3A 2A7

K. Fukuda Graduate School of Systems Management, University of Tsukuda, Tokyo, 3-29-1 Otsuka, Bunkyo-ku, Tokyo 112, Japan (Received 9 September 1993)

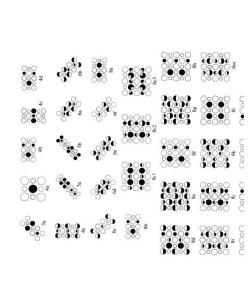
The possible ground states of a ternary fice lattice model with nearest- and next-nearest-neighbor pair interactions are investigated by constructing an eight-dimensional configuration polytope and enumerating its vertices. Although a structure could not be constructed for most of the vertices, 31 ternary ground states are found, some of which correspond to structures that have been observed experimentally.

php 

large problems. The drawbuck of the method is that among duplicates of the same vertex can be generated want duplicates of the same vertex can be generated when degenerary is present. While both methods successfully generated all revites of the double of description method seems to be more appropriate for his of comparation because of the high degenerary and St moderate size of the inequality system. For large system the only taskink algorithm for vertex estimated may become the form by tasking algorithm for vertex sentmentation.

#### III. RESULTS

The ground-state polytope we found is highly degenerate and consists of 4882 vertices in the eightdimensional space spanned by the correlation functions. Some of the vertices dound correspond to structures that can be transformed into each other by permutations of the La. and e. general. If these are considered to be the same structure, the total number of distinct structures is



SS = 4 = 1

Parallel Reverse Search		
Keverse Search	7 TIV	
Vertex Enumeration	V	NULL SECTION.
Outline		

Parallel Reverse Search	
Reverse Search	MIT problem
Vertex Enumeration	Case study:
Outline	

Polytope mit defined by 729 inequalities in 8 dimensions

Parallel Reverse Search		
Reverse Search		Menopolem
vertex Enumeration	- (	.VELLAV.
Outiline		

- Polytope mit defined by 729 inequalities in 8 dimensions
  Output consists of 4862 vertices

- Polytope mit defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
  In 1993 it took about 3 weeks to solve by cdd and 6 weeks by Irs (20MHz?)

	Outline	Vertex Enumeration	Reverse Search	Parallel Reverse Search
H (				

- Polytope mit defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by cdd and 6 weeks by Irs (20MHz?)
  - Goal: Goal: parallelize *Irs* for multicore workstations using existing code

- Polytope mit defined by 729 inequalities in 8 dimensions
  - Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by cdd and 6 weeks by Irs (20MHz?)
- Goal: Goal: parallelize *Irs* for multicore workstations using existing code
  - In 2012:

cddr+	lrs			mp	mplrs		
		cores=8	8=9	cores	cores=16	cores=32	=32
secs	secs	secs	sn	secs	sn	secs	sn
368	496	66	5.0	44	11.2	26	19

Table: mai64: Opteron 6272, 2.1GHz, 64 cores, speedups(su) on Irs

- Polytope mit defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by cdd and 6 weeks by Irs (20MHz?)
- Goal: Goal: parallelize *Irs* for multicore workstations using existing code
  - In 2012:

_			_
Irs	=32	sn	19
	cores=32	secs	26
	cores=16	sn	11.2
mplrs	cores	secs	44
	8=9	sn	5.0
	cores=8	secs	66
Irs		secs	496
cddr+		secs	368

Table: mai64: Opteron 6272, 2.1GHz, 64 cores, speedups(su) on Irs

- 32-core speedup of *plrs* on 1993 *mplrs*: about 140,000 times! (processor= $110 \times 1300$ =software)

### More cores

Name	lrs			mplrs se	secs/efficiency		
	(mai20)	96 cores	128 cores	160 cores	192 cores	256 cores	312 cores
c40	10002	329	247	203	179	134	129
	П	.48	.48	.46	.44	(.44)	(.37)
perm10	2381	115	94	85	96	64	61
	1	.34	.31	.28	.20	(.23)	(.20)
mit71	21920	989	516	412	350	231	205
	П	.54	.54	.54	.53	(09.)	(.55)
bv7	9040	302	229	184	158	86	88
		.49	.49	.49	.47	(.57)	(.52)
9d5	1774681	26700	43455	34457	28634	18657	15995
	1	.63	.62	.63	.63	(.72)	(69.)

Table: efficiency = speedup/number of cores (mai cluster)

### Even more cores ...

2			mplrs		
	1 core	300 cores	600 cores	900 cores	1200 cores
c40	17755	89	49	43	44
	1	99.	.60	.46	.34
mit71	36198	147	80	63	49
	Н	.82	.75	.64	.62
2vq	10594	48	27	27	29
	Н	.73	.05	.44	.30
905	2400648	9640	4887	3278	2570
	Н	.83	.82	.81	.78

Table: Tsubame2.5 at Tokyo Institute of Technology: secs/efficiency

Outline Vertex Enumeration Reverse Search Parallel Reverse Search Reverse Search (A. & Fukuda, '91)

Space efficient technique to list unstructured discrete objects

Parallel Keverse Search	1)
Keverse Search	A & Firkinda '9
Vertex Enumeration	Reverse Search (A. & Firklida, 191)
Outline	

- Space efficient technique to list unstructured discrete objects
  Typical Problems:

raiallel Ivevelse Sealell	
	'91
Neverse Sealon	Reverse Search (A. & Fukuda, '91)
	rch (
Vertex Enumeration	Reverse Sea

- Space efficient technique to list unstructured discrete objects
  Typical Problems:
  Generate all triangulations on a given point set.

Farallel Keverse Search	
Reverse Search	& Firkinda '91
tion	Reverse Search (A & Firklida
vertex Enumera	Reverse
Outiline	

- Space efficient technique to list unstructured discrete objects
  - Typical Problems:
- Generate all triangulations on a given point set.
  Generate all planar spanning tress on a given set of points.

php

Parallel Reverse Search	
Keverse Search	Reverse Search (A. & Fukuda, '91)
eration	Search (A.
Vertex Enum	Reverse
Outline	

- Space efficient technique to list unstructured discrete objects
- Typical Problems:

- Generate all triangulations on a given point set.
  Generate all planar spanning tress on a given set of points.
  Generate all the cells or vertices of an arrangement of lines planes, or hyperplanes.

raidiei Neverse Searcii	
Neverse Search	Reverse Search (A. & Fukuda. '91`
	Search (A. &
verex Enumerate	Reverse

- Space efficient technique to list unstructured discrete objects
- Typical Problems:

- Generate all triangulations on a given point set.
  Generate all planar spanning tress on a given set of points.
  Generate all the cells or vertices of an arrangement of lines planes, or hyperplanes.

  Generate all vertices of a convex polyhedron

raidiei Neverse Searcii	
Neverse Search	Reverse Search (A. & Fukuda. '91`
	Search (A. &
verex Enumerate	Reverse

- Space efficient technique to list unstructured discrete objects
- Typical Problems:

- Generate all triangulations on a given point set.
  Generate all planar spanning tress on a given set of points.
  Generate all the cells or vertices of an arrangement of lines planes, or hyperplanes.

  Generate all vertices of a convex polyhedron
- Reverse search is defined by an adjacency oracle and a local search function



Parallel Reverse Search	
Reverse Search	Adjacency Oracle
Vertex Enumeration	Reverse Search -
Outline	

V are the objects to be generated



Parallel Keverse Search		
	-	Cracle
Keverse Search	<	ACIACENCY
Enumeration	-	Reverse Jearch - Adjacency Oracle
Vertex E		APVP PVPV
Outline		

- V are the objects to be generated Define graph G=(V,E) by:

V are the objects to be generated
Define graph G = (V, E) by:
For every v ∈ V i = 1, 2, ..., ∆ (maximum degree)

$$Adj(
u,i) = egin{cases} 
u' & ext{where } 
u' \in E \ 
otherwise 
otherwise$$

V are the objects to be generated

• Define graph G=(V,E) by: • For every  $v\in V$   $i=1,2,..,\Delta$  (maximum degree)

$$Adj(
u,i) = egin{cases} 
u' & ext{where } 
uv' \in E \ 
o ext{otherwise} 
onumber 
onumber$$

• For every edge w' in G there is a unique i such that v' = Adj(v,i).

V are the objects to be generated

• Define graph G=(V,E) by: • For every  $v\in V$   $i=1,2,..,\Delta$  (maximum degree)

$$Adj(
u,i) = egin{cases} 
u' & ext{where } 
u' \in E \ 
otherwise 
otherwise$$

• For every edge w' in G there is a unique i such that v' = Adj(v,i).

• "Similar" objects are joined by an edge

V are the objects to be generated

• Define graph G=(V,E) by: • For every  $v\in V$   $i=1,2,..,\Delta$  (maximum degree)

$$Adj(v,i) = egin{cases} v' & ext{where } vv' \in E \ arphi & ext{otherwise} \end{cases}$$

• For every edge vv' in G there is a unique i such that v' = Adj(v,i).

• "Similar" objects are joined by an edge

 $\bullet$  Maximum degree  $\Delta$  should be as small as possible

Parallel Reverse Search		
Reverse Search		ı - Local Search
Vertex Enumeration	(	Reverse Search -
Outline		

ullet G=(V,E) is the given graph

מומונו וענאני מען מעוני	
	th - Local Search
	Reverse Search -

- G=(V,E) is the given graph  $u^*\in V$  is a target vertex

- G = (V, E) is the given graph
  v\* ∈ V is a target vertex
  f: V → V is a local search function s.t.:

G = (V, E) is the given graph
v\* ∈ V is a target vertex
f: V → V is a local search function s.t.:
f(v\*) = v\*

G = (V, E) is the given graph
v\* ∈ V is a target vertex
f: V → V is a local search function s.t.:
f(v\*) = v\*
lterating f on any v leads to v\*

- G = (V, E) is the given graph
  v\* ∈ V is a target vertex
  f: V → V is a local search function s.t.:
  f(v\*) = v\*
  lterating f on any v leads to v\*
  le. f(f(f..(f(v))..) = v\*

ullet G=(V,E) is the given graph

v\* ∈ V is a target vertex
f: V → V is a local search function s.t.:
f(v\*) = v\*
lterating f on any v leads to v\*
le. f(f(f..(f(v))..) = v\*

- f defines a spanning tree on G rooted at  $\nu^*$ 

Reverse Search - Local Search

- ullet G=(V,E) is the given graph
- v\* ∈ V is a target vertex
  f: V → V is a local search function s.t.:
  f(v\*) = v\*
  lterating f on any v leads to v\*
  le. f(f(f..(f(v))..) = v\*

- f defines a spanning tree on  ${\cal G}$  rooted at  $\nu^*$
- $\bullet$  Reverse search generates this tree starting at  $\nu^{\ast}$

### Example - Problem

#### Problem:

Generate permutations of  $\{1,2,..,n\}$ 

#### Input:

n = 4

#### Output:

```
(1, 2, 3, 4) (1, 2, 4, 3) (1, 3, 2, 4) (1, 3, 4, 2) (1, 4, 2, 3) (1, 4, 3, 3) (2, 1, 3, 4) (2, 1, 4, 3) (2, 3, 1, 4) (2, 3, 4, 1) (2, 4, 1, 3) (2, 4, 3, 1) (3, 1, 2, 4) (3, 1, 4, 2) (3, 2, 1, 4) (3, 2, 4, 1) (3, 4, 1, 2) (3, 4, 2, 1) (4, 1, 2, 3) (4, 1, 3, 2) (4, 2, 1, 3) (4, 2, 3, 1) (4, 3, 1, 2) (4, 3, 2, 1)
```

$$(3,1,2,4)(3,1,4,2)(3,2,1,4)(3,2,4,1)(3,4,1,2)(3,4,$$

$$(4,1,2,3)(4,1,3,2)(4,2,1,3)(4,2,3,1)(4,3,1,2)(4,3,2,3,1)$$

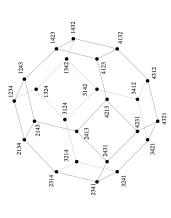
php 

Example - Adjacency Oracle

 $\{\pi_1,\pi_2,...,\pi_n\} \textit{isapermutationof}\, \{1,2,...,n\}$ 

 $Adj(\pi, i) = (\pi_1, \pi_2, ..., \pi_{i-1}, \pi_{i+1}, \pi_i, ...\pi_n)$  for i = 1, 2, ..., n-1.

Note:  $\Delta = n - 1$ 

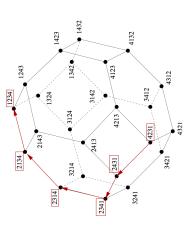


### Example - Local Search

Let  $\pi = (\pi_1, \pi_2, ..., \pi_n)$ Target: (1, 2, ..., n)

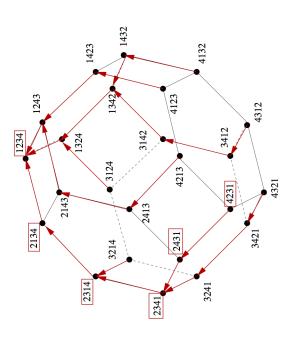
$$f(\pi) = (\pi_1, \pi_2, ..., \pi_{i-1}, \pi_{i+1}, \pi_i, ..., \pi_n)$$

where i is the smallest index for which  $\pi_i > \pi_{i+1}.$ 



Outline Vertex Enumeration Reverse Search Parallel Reverse Search

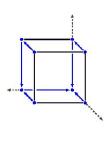
Example - Reverse Search Tree



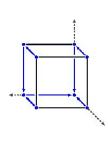
## Reverse Search - Pseudocode

```
backtrack step
                                                                                                          forward step
Algorithm 1 reverseSearch(v^*, \Delta, Adj, f)
             repeat v \leftarrow v^* j \leftarrow 0 while j < \Delta do j \leftarrow j + 1 if f(Adj(v,j)) = v then v \leftarrow Adj(v,j) print v \neq 0 end if end while if v \neq v^* then (v,j) \leftarrow f(v) end if
                                                                                                                                                                                                                                                       until v = v^* and j = \Delta
```

Parallel Reverse Search	on-l
Reverse Search	vertex enumerati
Vertex Enumeration	Reverse search for vertex enumeration-
Outiline	

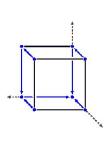


Reverse search for vertex enumeration-I



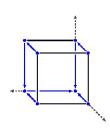
• G = (V, E) is defined by the vertices and edges of the polytope

Reverse search for vertex enumeration-I



- ullet  $G=\left(V,E
  ight)$  is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle

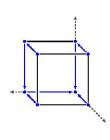
# Reverse search for vertex enumeration-I



- G=(V,E) is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle
- Simplex method gives a path from any vertex to the optimum vertex

INTERPORTED TO THE PORT OF THE

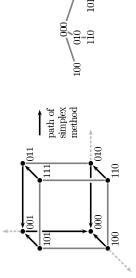
## Reverse search for vertex enumeration-I



- ullet  $G=\left(V,E
  ight)$  is defined by the vertices and edges of the polytope
  - Pivoting between vertices defines the adjacency oracle
- Simplex method gives a path from any vertex to the optimum vertex
- Irs is a C implementation available on-line

Reverse Search

# Reverse search for vertex enumeration-II http://cgm.cs.mcgill.ca/ avis/C/lrs.html



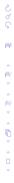
(a) The "simplex tree" induced by the objective  $(-\sum x_i)$ .

(b) The corresponding reverse search tree.

500 H 

raiallel Nevelse Sealell	ation
Nevelse Sealcii	Reverse Search: features for parallelization
Vertex Enameration	Reverse Search:
Outille	

• Objects generated are not stored in a database: no collisions



ation
features for parallelization
Reverse Search: fea

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards



	zation
בייייייייייייייייייייייייייייייייייייי	Reverse Search: features for parallelization
	Reverse Search: f

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
- Subtrees may be enumerated independently without communication



ratallet ivevelse Sealett	ization
cal C.	parallel
Deal Seal Seal Seal Seal Seal Seal Seal S	Reverse Search: features for parallelization
rex Enumeration	Search:
מוסא	Reverse
Ontille	

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
- Subtrees may be enumerated independently without communication
- Subtree size may be estimated by Hall-Knuth estimator



Outline Vertex Enumeration Reverse Search Parallel Reverse Search Extended Reverse Search

Extension to allow:

Extended Reverse Search

Extension to allow:

• all subtrees to be listed at some fixed depth

Parallel Reverse Search	
Reverse Search	Extended Reverse Search
Vertex Enumeration	Extended
Outiline	

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root



Extended Reverse Search
Extended R

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given rootAdditional parameters:



rarallel Neverse Search		
Neverse Searcii	-	Extended Keverse Search
Vertex Enumeration		Extended
Outillie		

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
  Additional parameters:
- maxd is the depth at which forward steps are terminated.



## Extended Reverse Search

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
  - Additional parameters:
- maxd is the depth at which forward steps are terminated.
  mind is the depth at which backtrack steps are terminated.

## Extended Reverse Search

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
  - Additional parameters:
- maxd is the depth at which forward steps are terminated.
  mind is the depth at which backtrack steps are terminated.
  d is the depth of subtree root v\*.

Reverse Se

\_

# Extended Reverse Search - Pseudocode

```
Algorithm 2 extendedReverseSearch(v^*, \Delta, Adj, f, d, maxd, mind)
```

```
repeat v \leftarrow v^* \ j \leftarrow 0 \mathbf{while} \ j < \Delta \ \text{and} \ d < maxd \ \mathbf{do} j \leftarrow j+1 \mathbf{if} \ f(Adj(v,j)) = v \ \mathbf{then} v \leftarrow Adj(v,j) \mathbf{print} \ v \\ j \leftarrow 0 d \leftarrow d+1 \mathbf{end} \ \mathbf{if} \mathbf{end} \ \mathbf{if} \mathbf{end} \ \mathbf{while} \mathbf{if} \ v \neq v^* \ \mathbf{then} (v,j) \leftarrow f(v) d \leftarrow d-1 \mathbf{end} \ \mathbf{if} \mathbf{until} \ (d = mind \ or \ v = v^*) \ \mathbf{and} \ j = \Delta
```

Parallel Reverse Search	
Reverse Search	n design parameters
Vertex Enumeration	Parallelization
Outline	

rarallel Neverse Searcil	
verse Search	n parameters
ב	desig
Vertex Enumeration	Parallelization design parameters
Outillie	

• Users are from many disciplines and are not software engineers!



raialiei Nevelse Sealcii	
Nevelse Sealcii	design parameters
Vertex Enumeration	Parallelization design parameters
Outilie	

- Users are from many disciplines and are not software engineers!
- No special setup, extra library installation, or change of usage for users



raiallel Nevelse Sealell		
Neverse Search		Farallelization design parameters
Vertex Elimileration	D	Farallelization
Outillie		

- Users are from many disciplines and are not software engineers!
- No special setup, extra library installation, or change of usage for users
- Use available cores on user machine 'automatically'



Parallel Reverse Search	
Keverse Search	design parameters
Vertex Enumeration	Parallelization of
Outline	

# Users are from many disciplines and are not software engineers!

- No special setup, extra library installation, or change of usage for users
- Use available cores on user machine 'automatically'
- Reuse existing *Irs* code (8,000+ lines!)



_	
	3 phases
	Naive Parallel Reverse Search: 3 phases
	Reverse
	Parallel
	Naive

- Phase 1: (single processor)
- Generate the reverse search tree T down to a fixed depth init\_depth.
   Redirect output nodes and store in list L.

pop

_	
arailei Keverse Searc	
Ľ	3 phases
Reverse Search	Naive Parallel Reverse Search: 3 phases
	Reverse
ex Enumeration	Parallel
vert	Naive
Outiline	

- Phase 1: (single processor)
- $\bullet$  Generate the reverse search tree  ${\cal T}$  down to a fixed depth init\_depth.
  - Redirect output nodes and store in list *L*.
- Phase 2: (full parallelization)
- Schedule threads from L using subtree enumeration feature.
  Use parameter max\_threads to limit number of parallel threads.
  Direct output to shared output stream.



arcn	
arailei Keverse Se	
L	3 phases
Reverse Search	Vaive Parallel Reverse Search: 3 phases
	Reverse
ex Enumeration	Parallel
ver	Naive
Outiline	

- Phase 1: (single processor)
- Generate the reverse search tree  ${\cal T}$  down to a fixed depth init\_depth.
  - Redirect output nodes and store in list L.
- Phase 2: (full parallelization)
- Schedule threads from L using subtree enumeration feature.
- Use parameter max\_threads to limit number of parallel threads.
  - Direct output to shared output stream.
- Phase 3: (partial parallelization)
- Wait until all children threads terminate.



Vertex Enumeration Reverse Search Parallel Reverse Search

## Parallel Reverse Search - Pseudocode

```
extendedReverseSearch(\nu, \Delta, Adj, f, depth(\nu), \infty, depth(\nu))
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          extendedReverseSearch(v, \Delta, Adj, f, depth(v), \infty, depth(v))
                                                                        Phase 1
                                                                                                                                                                                                                                 Phase 2
Algorithm 3 parallelReverseSearch(v^*, \Delta, Adj, f, id, mt)
                                                                                                                                     remove all v \in L with depth(v) < id and output v
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           wait until a termination signal is received
                                                                                                         extendedReverseSearch(v^*, \Delta, Adj, f, 0, id, 0)
                                                                                                                                                                                                                                        remove any v \in \mathcal{L} num\_threads \leftarrow num\_threads + 1
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         num\_threads \leftarrow num\_threads - 1
                                                                                                                                                                                                       if num_threads < mt then</pre>
                                                                                                                                                                                                                                                                                                                                                                                              while num\_threads > 0 do wait for termination signal
                                                                        redirect output to a list L
                                              num\_threads \leftarrow 0
                                                                                                                                                                                                                                                                                                                                                                                                                                                           if L \neq \emptyset then
                                                                                                                                                                      while L \neq \varnothing do
                                                                                                                                                                                                                                                                                                                                                                  end while
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    end while
                                                                                                                                                                                                                                                                                                                                   end if
```

A portable parallel implementation of Irs derived from the parallel reverse search algorithm.

#### Architecture:

- Light C++ wrapper around *lrs*.
- Leverage Irs's restart feature.
  Use portable g++ compiler.
- Multi-producer and single consumer.
- Producer threads traverse subtrees of the reverse search tree,
  - appending nodes to a lock-free queue.

    Consumer thread removes nodes from shared queue and concatenates to unified location.
- Leverage open source Boost library for atomic features.
- Ensures portability, maintainability and strong performance.

3 Phases: CPU utilization

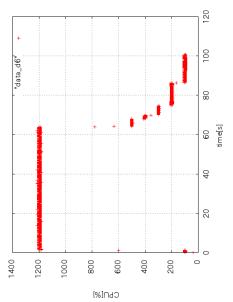
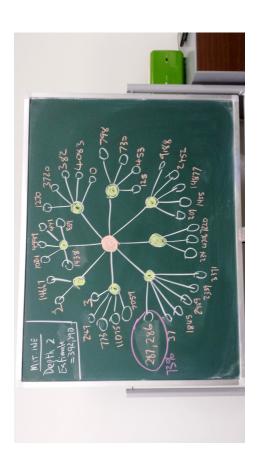
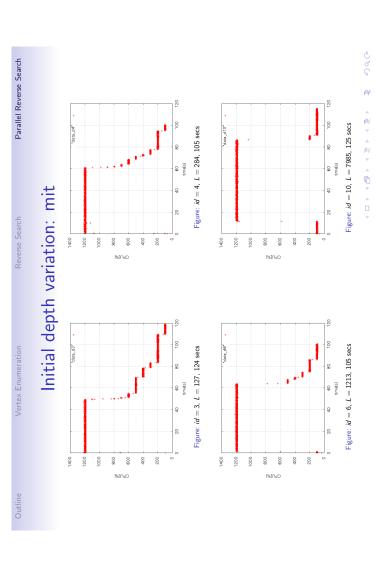


Figure: Input file: mit, id = 6, cores=12

Vertex Enumeration Reverse Search Parallel Reverse Search Estimates at depth 2: mit





plrs: limitations
plrs:

### Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.

	plrs: limitations
	plrs:
)	

### Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.
  - This leads to the following issues:

	plrs: limitations
	plrs:
)	

- Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.
  - This leads to the following issues:
- Success depends on balance of the reverse search tree.

Parallel Reverse Search			
Keverse Search			
Vertex Enumeration			
Outline			

### plrs: limitations

- Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.
  - This leads to the following issues:
- Success depends on balance of the reverse search tree.
   Conflicting issues in setting init\_depth.

Parallel Reverse Search			
Keverse Search			
Vertex Enumeration			
Outline			

### plrs: limitations

- Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.
  - This leads to the following issues:
- Success depends on balance of the reverse search tree.
  Conflicting issues in setting *init\_depth*.
  These problems were solved in mplrs

Parallel Reverse Search			
Keverse Search			
Vertex Enumeration			
Outline			

#### plrs: limitations

- Algorithm analysis:

- No parallelization in Phase 1.
  Complete parallelizatin in Phase 2.
  Parallelization drops monotonically in Phase 3.
  - This leads to the following issues:
- Success depends on balance of the reverse search tree.
  Conflicting issues in setting *init\_depth*.
  These problems were solved in mplrs
  Please come back for part 2!