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

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.

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.

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.

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

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

The decision variables are $x_1, x_2, ..., x_6$.

Fractional servings are allowed.

From *Linear Programming*, Vasek Chvátal, 1983

Linear programming formulation for diet problem

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

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

Linear programming solution

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

Linear programming solution

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

- $x_1 = 4$(oatmeal) $x_4 = 4.5$(milk) $x_5 = 2$(pie) cost=92.5 ¢

Linear programming solution

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

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

Linear programming solution

	Food	Serv. Size	Energy (kcal)	Protein (g)	Calcium (mg)	Price ¢	Max Serv.
x_1	Oatmeal	28g	110	4	2	3	4
x_2	Chicken	100g	205	32	12	24	3
x_3	Eggs	2 large	160	13	54	13	2
x_4	Milk	237ml	160	8	285	9	8
x_5	Cherry Pie	170g	420	4	22	20	2
x_6	Pork w. beans	260g	260	14	80	19	2
	Min. Daily Amt.		2000	55	800		

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

Problems with the solution

Problems with the solution

- Many desirable items were not included in the optimum solution

Problems with the solution

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

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!

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!
- Ask the right question !

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!
- Ask the right question !
- What are all the meals **I** can eat for at most $1?

All meals for a dollar

Replace the objective function by an inequality:

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

$$\begin{aligned}
110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 &\geq 2000 \\
4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 &\geq 55 \\
2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 &\geq 800 \\
0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \\
0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2
\end{aligned}$$

All meals for a dollar

Replace the objective function by an inequality:

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

$$\begin{aligned}
110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 &\geq 2000\\
4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 &\geq 55\\
2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 &\geq 800\\
0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2,&\\
0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2&
\end{aligned}$$

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

All meals for a dollar

Replace the objective function by an inequality:

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

$$\begin{aligned}
110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 &\geq 2000 \\
4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 &\geq 55 \\
2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 &\geq 800 \\
0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2,& \\
0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2&
\end{aligned}$$

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

All meals for a dollar

Replace the objective function by an inequality:

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

$$\begin{aligned}
110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 &\geq 2000 \\
4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 &\geq 55 \\
2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 &\geq 800 \\
0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2,& \\
0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2&
\end{aligned}$$

- 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

All menus for a $1

All (17) Extreme

Solutions to the Diet Problem with Budget $1.00

Cost	Oat-meal	Chicken	Eggs	Milk	Cherry Pie	Pork Beans
92.5	4.	0	0	4.5	2.	0
97.3	4.	0	0	8.	0.67	0
98.6	4.	0	0	2.23	2.	1.40
100.	1.65	0	0	6.12	2.	0
100.	2.81	0	0	8.	0.98	0
100.	3.74	0	0	2.20	2.	1.53
100.	4.	0	0	2.18	1.88	1.62
100.	4.	0	0	2.21	2.	1.48
100.	4.	0	0	5.33	2.	0
100.	4.	0	0	8.	0.42	0.40
100.	4.	0	0	8.	0.80	0
100.	4.	0	0.50	8.	0.48	0
100.	4.	0	1.88	2.63	2.	0
100.	4.	0.17	0	2.27	2.	1.24
100.	4.	0.19	0	8.	0.58	0
100.	4.	0.60	0	3.73	2.	0
100.	4.	0	1.03	2.21	2.	0.78

A more useful solution

All menus for a \$1

All (17) Extreme
Solutions to the Diet Problem with Budget \$1.00

Cost	Oatmeal	Chicken	Eggs	Milk	Cherry Pie	Pork Beans
92.5	4.	0	0	4.5	2.	0
97.3	4.	0	0	8.	0.67	0
98.6	4.	0	0	2.23	2.	1.40
100.	1.65	0	0	6.12	2.	0
100.	2.81	0	0	8.	0.98	0
100.	3.74	0	0	2.20	2.	1.53
100.	4.	0	0	2.18	1.88	1.62
100.	4.	0	0	2.21	2.	1.48
100.	4.	0	0	5.33	2.	0
100.	4.	0	0	8.	0.42	0.40
100.	4.	0	0	8.	0.80	0
100.	4.	0	0.50	8.	0.48	0
100.	4.	0	1.88	2.63	2.	0
100.	4.	0.17	0	2.27	2.	1.24
100.	4.	0.19	0	8.	0.58	0
100.	4.	0.60	0	3.73	2.	0
100.	4.	0	1.03	2.21	2.	0.78

A more useful solution

All menus for a $1

All (17) Extreme
Solutions to the Diet Problem with Budget $1.00

Cost	Oat-meal	Chicken	Eggs	Milk	Cherry Pie	Pork Beans
92.5	4.	0	0	4.5	2.	0
97.3	4.	0	0	8.	0.67	0
98.6	4.	0	0	2.23	2.	1.40
100.	1.65	0	0	6.12	2.	0
100.	2.81	0	0	8.	0.98	0
100.	3.74	0	0	2.20	2.	1.53
100.	4.	0	0	2.18	1.88	1.62
100.	4.	0	0	2.21	2.	1.48
100.	4.	0	0	5.33	2.	0
100.	4.	0	0	8.	0.42	0.40
100.	4.	0	0	8.	0.80	0
100.	4.	0	0.50	8.	0.48	0
100.	4.	0	1.88	2.63	2.	0
100.	4.	0.17	0	2.27	2.	1.24
100.	4.	0.19	0	8.	0.58	0
100.	4.	0.60	0	3.73	2.	0
100.	4.	0	1.03	2.21	2.	0.78

- Taking convex combinations of rows gives new meals

A more useful solution

All menus for a $1

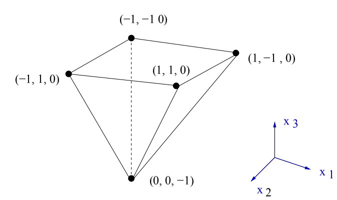
All (17) Extreme

Solutions to the Diet Problem with Budget $1.00

Cost	Oat-meal	Chicken	Eggs	Milk	Cherry Pie	Pork Beans
92.5	4.	0	0	4.5	2.	0
97.3	4.	0	0	8.	0.67	0
98.6	4.	0	0	2.23	2.	1.40
100.	1.65	0	0	6.12	2.	0
100.	2.81	0	0	8.	0.98	0
100.	3.74	0	0	2.20	2.	1.53
100.	4.	0	0	2.18	1.88	1.62
100.	4.	0	0	2.21	2.	1.48
100.	4.	0	0	5.33	2.	0
100.	4.	0	0	8.	0.42	0.40
100.	4.	0	0	8.	0.80	0
100.	4.	0	0.50	8.	0.48	0
100.	4.	0	1.88	2.63	2.	0
100.	4.	0.17	0	2.27	2.	1.24
100.	4.	0.19	0	8.	0.58	0
100.	4.	0.60	0	3.73	2.	0
100.	4.	0	1.03	2.21	2.	0.78

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

Example in R^3



H-representation:

$$
\begin{aligned}
1 - x_1 \quad\quad + x_3 &\ge 0\\
1 \quad\quad - x_2 + x_3 &\ge 0\\
1 + x_1 \quad\quad + x_3 &\ge 0\\
1 \quad\quad + x_2 + x_3 &\ge 0\\
- x_3 &\ge 0
\end{aligned}
$$

V-representation:

$v_1 = (-1, 1, 0)$, $v_2 = (-1, -1, 0)$, $v_3 = (1, -1, 0)$,

$v_4 = (1, 1, 0)$, $v_5 = (0, 0, -1)$

Two representations of a bounded polyhedron

Two representations of a bounded polyhedron

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

Two representations of a bounded polyhedron

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

$$
\begin{aligned}
x &= \sum_{i=1}^{N} \lambda_i v_i \\
\text{where} \quad \sum_{i=1}^{N} \lambda_i &= 1, \quad \lambda_i \geq 0, \;\; i = 1, 2, ..., N
\end{aligned}
$$

Two representations of a bounded polyhedron

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

$$x = \sum_{i=1}^{N} \lambda_i v_i$$

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

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

Two representations of a bounded polyhedron

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

$$x = \sum_{i=1}^{N} \lambda_i v_i$$

$$\text{where } \sum_{i=1}^{N} \lambda_i = 1, \quad \lambda_i \geq 0, \; i = 1, 2, \ldots, N$$

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

Two representations of a bounded polyhedron

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

$$x = \sum_{i=1}^{N} \lambda_i v_i$$

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

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

Who uses vertex enumeration?

Who uses vertex enumeration?

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

Who uses vertex enumeration?

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

Who uses vertex enumeration?

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

Who uses vertex enumeration?

- 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 ...

Who uses vertex enumeration?

- 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 ...
- ... should run faster on better hardware!

Who uses vertex enumeration?

- 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 ...
- ... should run faster on better hardware!
- Goal: parallelize *lrs* for multicore workstations using existing code

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. Garbulsky
Department of Materials Science and Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

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

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

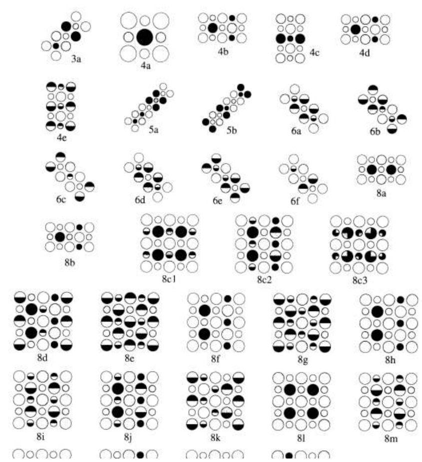
The possible ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor pair interactions are investigated by constructing an eight-dimensional configuration polytope and enumerating its vertices. Although a structure could not be constructed for most of the vertices, 31 ternary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

Case study: MIT problem

large problems. The drawback of the method is that many duplicates of the same vertex can be generated when degeneracy is present. While both methods successfully generated all vertices of the polytope, the double description method seems to be more appropriate for this computation because of the high degeneracy and moderate size of the inequality system. For larger systems, however, the reverse search method may become the only feasible algorithm for vertex enumeration.

III. RESULTS

The ground-state polytope we found is highly degenerate and consists of 4862 vertices in the eight-dimensional space spanned by the correlation functions. Some of the vertices found correspond to structures that can be transformed into each other by permutations of the A, B, and C species. If these are considered to be the same structure, the total number of distinct structures is



Case study: MIT problem

Case study: MIT problem

- Polytope mit defined by 729 inequalities in 8 dimensions

Case study: MIT problem

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

Case study: MIT problem

- 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 *lrs* (20MHz?)

Case study: MIT problem

- 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 *lrs* (20MHz?)
- Goal: Goal: parallelize *lrs* for multicore workstations using existing code

Case study: MIT problem

- 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 *lrs* (20MHz?)
- Goal: Goal: parallelize *lrs* for multicore workstations using existing code
- In 2012:

cddr+	*lrs*	*mplrs*					
		cores=8		cores=16		cores=32	
secs	secs	secs	su	secs	su	secs	su
368	496	99	5.0	44	11.2	26	19

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

Case study: MIT problem

- 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 *lrs* (20MHz?)
- Goal: Goal: parallelize *lrs* for multicore workstations using existing code
- In 2012:

cddr+	*lrs*	*mplrs*					
		cores=8		cores=16		cores=32	
secs	secs	secs	su	secs	su	secs	su
368	496	99	5.0	44	11.2	26	19

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

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

More cores

Name	lrs	mplrs	secs/efficiency				
	(mai20)	96 cores	128 cores	160 cores	192 cores	256 cores	312 cores
c40	10002	329	247	203	179	134	129
	1	.48	.48	.46	.44	(.44)	(.37)
perm10	2381	115	94	85	96	64	61
	1	.34	.31	.28	.20	(.23)	(.20)
mit71	21920	686	516	412	350	231	205
	1	.54	.54	.54	.53	(.60)	(.55)
bv7	9040	302	229	184	158	98	88
	1	.49	.49	.49	.47	(.57)	(.52)
cp6	1774681	56700	43455	34457	28634	18657	15995
	1	.63	.62	.63	.63	(.72)	(.69)

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

Even more cores ...

Name	mplrs				
	1 core	300 cores	600 cores	900 cores	1200 cores
c40	17755	89	49	43	44
	1	.66	.60	.46	.34
mit71	36198	147	80	63	49
	1	.82	.75	.64	.62
bv7	10594	48	27	27	29
	1	.73	.65	.44	.30
cp6	2400648	9640	4887	3278	2570
	1	.83	.82	.81	.78

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

Reverse Search (A. & Fukuda, '91)

- Space efficient technique to list unstructured discrete objects

Reverse Search (A. & Fukuda, '91)

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

Reverse Search (A. & Fukuda, '91)

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

Reverse Search (A. & Fukuda, '91)

- 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.

Reverse Search (A. & Fukuda, '91)

- 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.

Reverse Search (A. & Fukuda, '91)

- 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 (A. & Fukuda, '91)

- 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

Reverse Search - Adjacency Oracle

- V are the objects to be generated

Reverse Search - Adjacency Oracle

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

Reverse Search - Adjacency Oracle

- V are the objects to be generated
- Define graph $G = (V, E)$ by:
- For every $v \in V \ i = 1, 2, .., \Delta$ (*maximum degree*)

$$Adj(v,i) = \begin{cases} v' & \text{where } vv' \in E \\ \varnothing & \text{otherwise} \end{cases}$$

Reverse Search - Adjacency Oracle

- V are the objects to be generated
- Define graph $G = (V, E)$ by:
- For every $v \in V$ $i = 1, 2, .., \Delta$ (*maximum degree*)

$$Adj(v,i) = \begin{cases} v' & \text{where } vv' \in E \\ \varnothing & \text{otherwise} \end{cases}$$

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

Reverse Search - Adjacency Oracle

- V are the objects to be generated
- Define graph $G = (V, E)$ by:
- For every $v \in V$ $i = 1, 2, .., \Delta$ (*maximum degree*)

$$Adj(v, i) = \begin{cases} v' & \text{where } vv' \in E \\ \varnothing & \text{otherwise} \end{cases}$$

- For every edge vv' in G there is a unique i such that $v' = Adj(v, i)$.
- "Similar" objects are joined by an edge

Reverse Search - Adjacency Oracle

- V are the objects to be generated
- Define graph $G = (V, E)$ by:
- For every $v \in V$ $i = 1, 2, .., \Delta$ (*maximum degree*)

$$Adj(v, i) = \begin{cases} v' & \text{where } vv' \in E \\ \varnothing & \text{otherwise} \end{cases}$$

- For every edge vv' in G there is a unique i such that $v' = Adj(v, i)$.
- "Similar" objects are joined by an edge
- Maximum degree Δ should be as small as possible

Reverse Search - Local Search

- $G = (V, E)$ is the given graph

Reverse Search - Local Search

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

Reverse Search - Local Search

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

Reverse Search - Local Search

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

Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating f on any v leads to v^*

Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating f on any v leads to v^*
  - Ie. $f(f(f..(f(v))..) = v^*$

Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
    - $f(v^*) = v^*$
    - Iterating f on any v leads to v^*
    - Ie. $f(f(f..(f(v))..) = v^*$
- f defines a spanning tree on G rooted at v^*

Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating f on any v leads to v^*
  - Ie. $f(f(f..(f(v))..) = v^*$
- f defines a spanning tree on G rooted at v^*
- Reverse search generates this tree starting at v^*

Example - Problem

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

Input:
$n = 4$

Output:

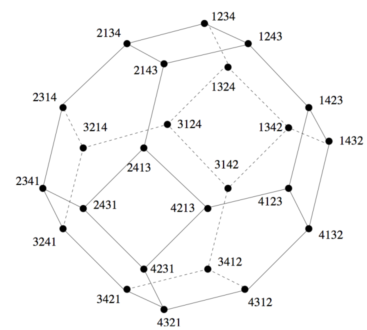
$$
\begin{array}{llllll}
(1,2,3,4) & (1,2,4,3) & (1,3,2,4) & (1,3,4,2) & (1,4,2,3) & (1,4,3,3) \\
(2,1,3,4) & (2,1,4,3) & (2,3,1,4) & (2,3,4,1) & (2,4,1,3) & (2,4,3,1) \\
(3,1,2,4) & (3,1,4,2) & (3,2,1,4) & (3,2,4,1) & (3,4,1,2) & (3,4,2,1) \\
(4,1,2,3) & (4,1,3,2) & (4,2,1,3) & (4,2,3,1) & (4,3,1,2) & (4,3,2,1)
\end{array}
$$

Example - Adjacency Oracle

$\{\pi_1, \pi_2, ..., \pi_n\} is a permutation of \{1, 2, .., n\}$

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

Note: $\Delta = n - 1$

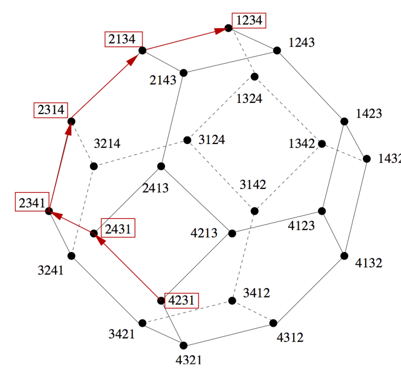


Example - Local Search

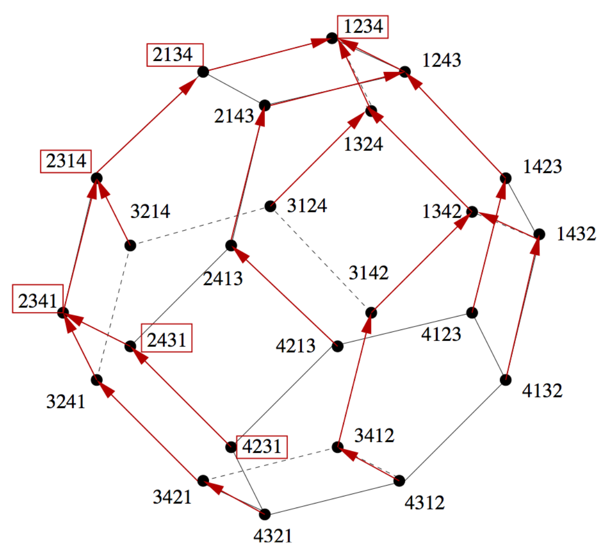
Let $\pi = (\pi_1, \pi_2, ..., \pi_n)$
Target: $(1, 2, ..., n)$

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

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



Example - Reverse Search Tree

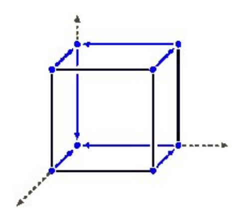


Reverse Search - Pseudocode

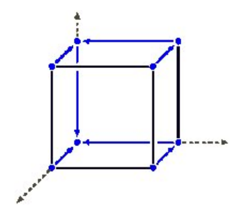
Algorithm 1 reverseSearch(v^*, Δ, Adj, f)

```
repeat
  v ← v*  j ← 0
  while j < Δ do
    j ← j + 1
    if f(Adj(v, j)) = v then
      v ← Adj(v, j)              forward step
      print  v
      j ← 0
    end if
  end while
  if  v ≠ v* then
    (v, j) ← f(v)                backtrack step
  end if
until v = v* and j = Δ
```

Reverse search for vertex enumeration-I

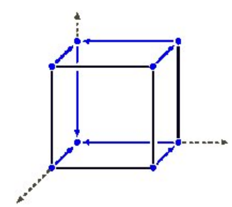


Reverse search for vertex enumeration-I



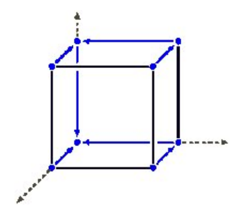
- $G = (V, E)$ is defined by the vertices and edges of the polytope

Reverse search for vertex enumeration-I



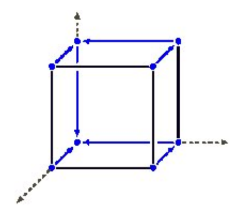
- $G = (V, E)$ is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle

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

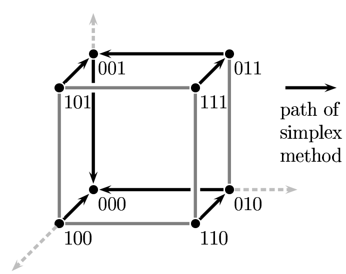
Reverse search for vertex enumeration-I



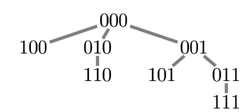
- $G = (V, E)$ is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle
- Simplex method gives a path from any vertex to the optimum vertex
- lrs is a C implementation available on-line

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.

Reverse Search: features for parallelization

Reverse Search: features for parallelization

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

Reverse Search: features for parallelization

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

Reverse Search: features for parallelization

- 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

Reverse Search: features for parallelization

- 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

Extended Reverse Search

Extension to allow :

Extended Reverse Search

Extension to allow :

- all subtrees to be listed at some fixed depth

Extended Reverse Search

Extension to allow :

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

Extended Reverse Search

Extension to allow :

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

Extended Reverse Search

Extension to allow :

- 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

Extension to allow :

- 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

Extension to allow :

- 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^*.

Extended Reverse Search - Pseudocode

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

```
repeat
  v ← v*  j ← 0
  while j < Δ and d < maxd do
    j ← j + 1
    if f(Adj(v, j)) = v then
      v ← Adj(v, j)                    forward step
      print v
      j ← 0
      d ← d + 1
    end if
  end while
  if v ≠ v* then
    (v, j) ← f(v)                      backtrack step
    d ← d − 1
  end if
until (d = mind or v = v*) and j = Δ
```

Parallelization design parameters

Parallelization design parameters

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

Parallelization design parameters

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

Parallelization design parameters

- 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'

Parallelization design parameters

- 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 *lrs* code (8,000+ lines!)

Naive Parallel Reverse Search: 3 phases

- 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.

Naive Parallel Reverse Search: 3 phases

- 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.
- 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.

Naive Parallel Reverse Search: 3 phases

- 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.
- 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.

Parallel Reverse Search - Pseudocode

Algorithm 3 parallelReverseSearch($v^*, \Delta, Adj, f, id, mt$)

```
    num_threads ← 0
    redirect output to a list L                                   Phase 1
    extendedReverseSearch(v*, Δ, Adj, f, 0, id, 0)
    remove all v ∈ L with depth(v) < id and output v
    while L ≠ ∅ do
      if num_threads < mt then
        remove any v ∈ L                                          Phase 2
        num_threads ← num_threads + 1
        extendedReverseSearch(v, Δ, Adj, f, depth(v), ∞, depth(v))
      end if
    end while
    while num_threads > 0 do
      wait for termination signal
      if L ≠ ∅ then
        wait until a termination signal is received
        extendedReverseSearch(v, Δ, Adj, f, depth(v), ∞, depth(v))
      else
        num_threads ← num_threads − 1                             Phase 3
      end if
    end while
```

plrs (Implemented by Gary Roumanis)

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

Architecture:

- Light C++ wrapper around *lrs*.
    - Leverage *lrs'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

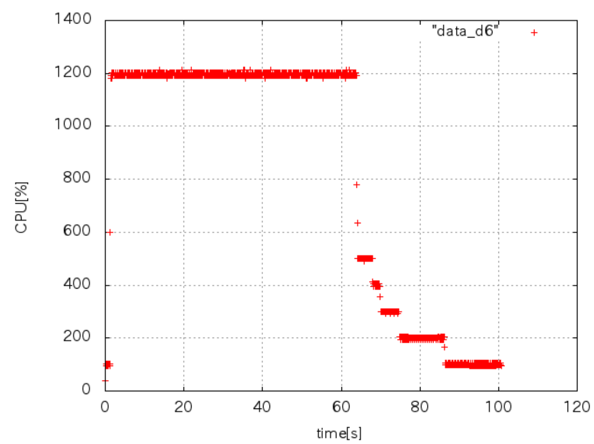
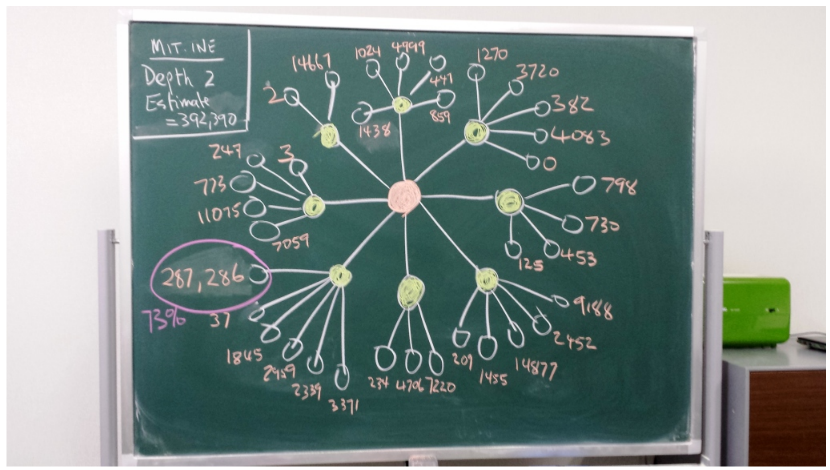


Figure: Input file: mit, $id = 6$, cores=12

Estimates at depth 2: mit



Initial depth variation: mit

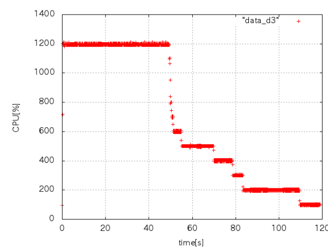


Figure: $id = 3$, $L = 127$, 124 secs

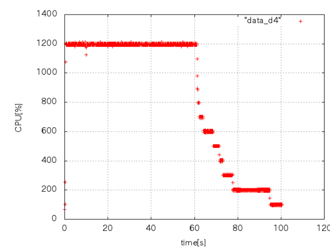


Figure: $id = 4$, $L = 284$, 105 secs

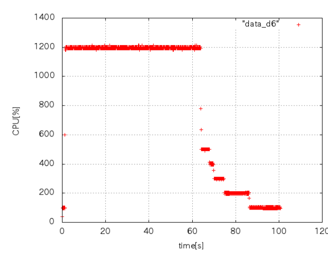


Figure: $id = 6$, $L = 1213$, 105 secs

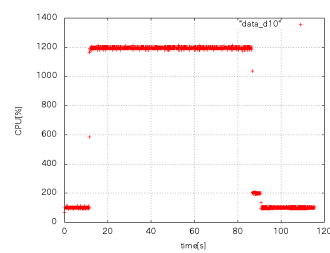


Figure: $id = 10$, $L = 7985$, 125 secs

plrs: limitations

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

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:

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.

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*.

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

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!