A New Generic Separation Algorithm and Its Application to the Vehicle Routing Problem

Presented by:

Ted Ralphs

Joint work with:

Leo Kopman

Les Trotter

Bill Pulleyblank

Outline of Talk

- Introduction
- Description of the Vehicle Routing Problem
- Description of the Decomposition Algorithm
- Extensions to the Basic Algorithm
- Implementation Details
- Computational Results
- Conclusions

Combinatorial Optimization

A *combinatorial optimization problem* $CP = (E, \mathcal{F})$ consists of

- A ground set E, and
- A set $\mathcal{F} \subseteq 2^E$ of *feasible solutions* to CP.

Given a cost function $c \in \mathbf{Z^E}$, we define the cost of $S \in \mathcal{F}$ to be $c(S) = \sum_{e \in S} c_e$. A *subproblem* is defined by $\mathcal{S} \subseteq \mathcal{F}$.

<u>Problem</u>: Find a least cost member of $\mathcal{F}$.

<u>Approach</u>: Apply branch and cut techniques.

Example: The Vehicle Routing Problem

The VRP is a combinatorial problem $(E, \mathcal{F})$ whose ground set is the edges of a graph $G(E, N)$. Notation:

- d is a vector of the demands.
- N is the set of customers plus the depot (node 0).
- $N^- = N \setminus \{0\}$.
- k is the number of routes.
- C is the capacity of a truck.

A feasible solution is composed of:

- a partition $\{R_1, \ldots, R_k\} \subseteq 2^N$ such that $\sum_{j \in R_i} d_j \leq C,\ 1 \leq i \leq k$;
- a permutation σ_i of $R_i \cup \{0\}$ specifying the order in which the customers on route i are to be serviced, $1 \leq i \leq k$.

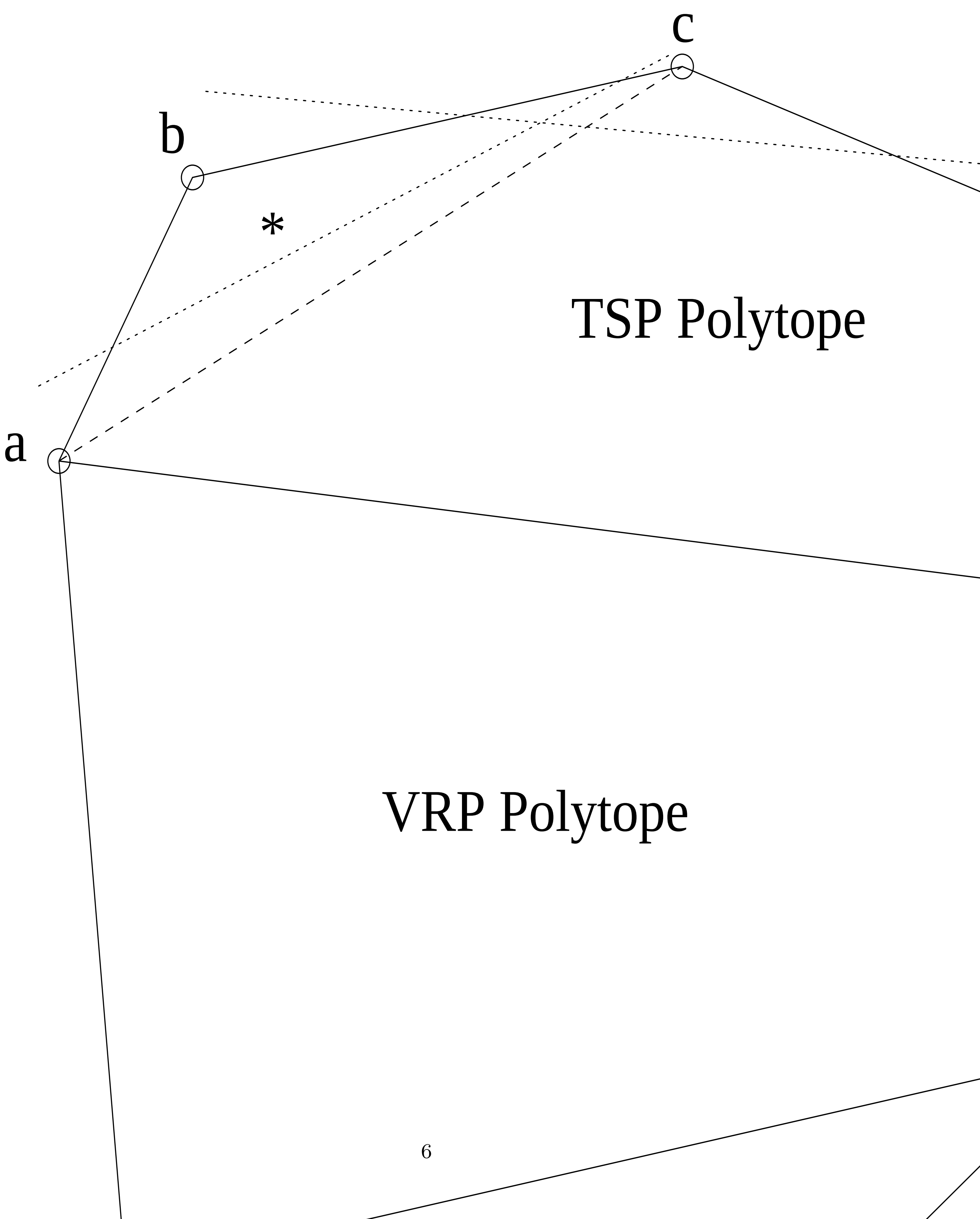
Feasible solutions are those incidence vectors satisfying:

$$\begin{array}{rcll} \sum_{j=1}^{n} x_{0j} & = & 2k & \\ \sum_{j=1}^{n} x_{ij} & = & 2 & \forall i \in N^- \\ \sum_{\substack{i \in S \\ j \notin S}} x_{ij} & \geq & 2b(S) & \forall S \subset N^-,\ |S| > 1. \end{array}$$

$b(S)$ = lower bound on the minimum number of trucks required to service the customers in S (usually $\lceil (\sum_{i \in S} d_i)/C \rceil$)

Motivation: Solving Combinatorial Problems with Side Constraints

- Feasible solutions to the VRP, called *routings*, can be thought of as TSP tours on a slightly expanded graph obtained by duplicating the depot $k-1$ times.
- The set of TSP tours on this graph that also obey the capacity constraints corresponds to routings for the corresponding VRP.
- Hence, the TSP can be thought of as a relaxation of the VRP.
- The capacity constraints constitute a family of *side constraints* for the TSP with respect to the VRP.
- In general, suppose we have two combinatorial problems, $CP = (E, \mathcal{F})$ and $CP' = (E, \mathcal{H})$ such that $\mathcal{H} \subseteq \mathcal{F}$. CP is then a relaxation of CP'.
- We call $ax \geq \beta$ a *side constraint* if it is a valid inequality for $CPP' = conv\Big(\{x^S : S \in \mathcal{H}\}\Big)$ but not for $CPP = conv\Big(\{x^S : S \in \mathcal{F}\}\Big)$.
- Key observation: We can determine in $O(n)$ time whether a particular TSP tour satisfies all the capacity constraints.


c
b
*
TSP Polytope
a
VRP Polytope

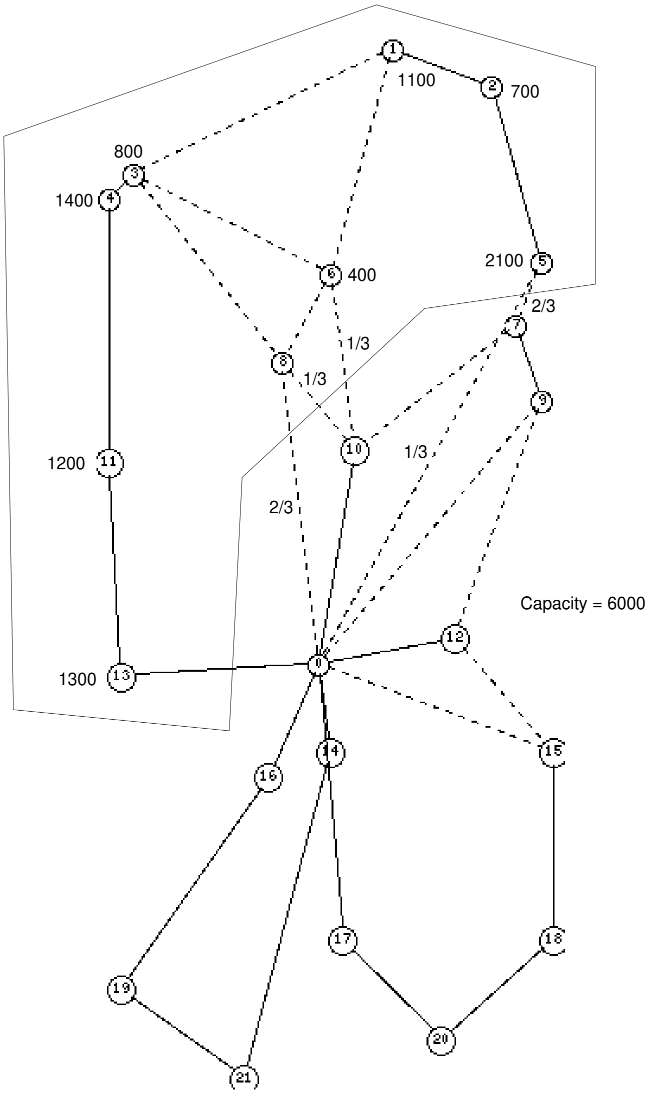
The Decomposition Algorithm

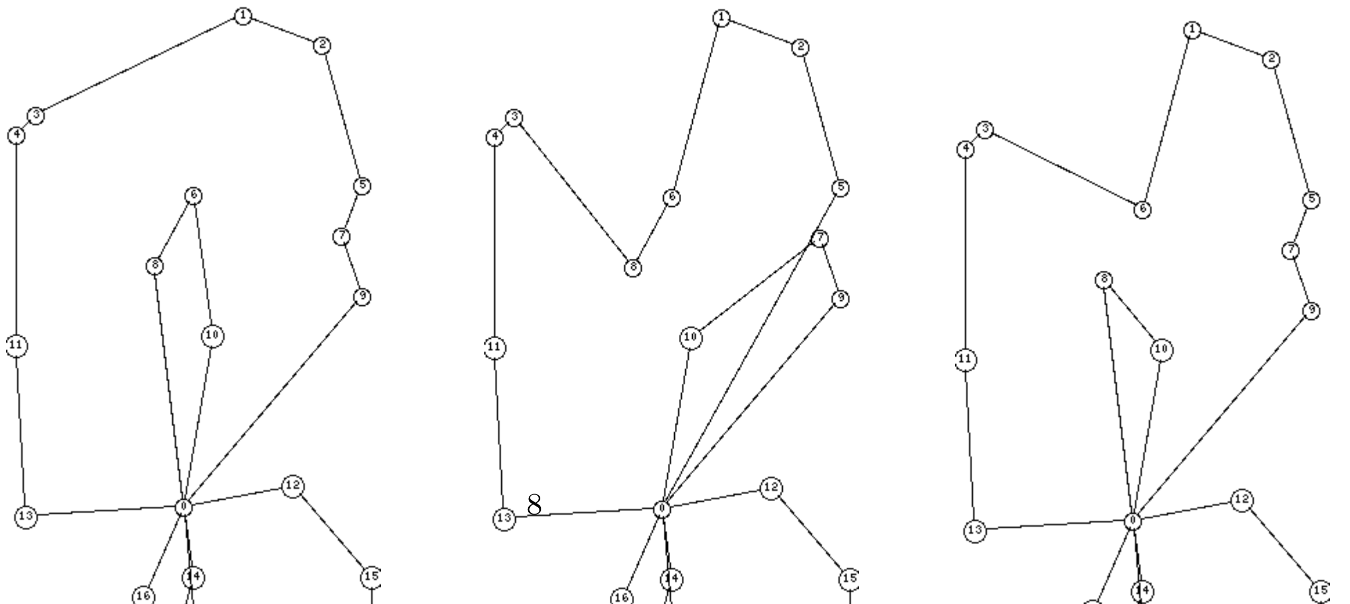
- Attempt to decompose the fractional solution $\hat{x}$ into a convex combination of extreme points of CPP by solving the *Decomposition LP* defined by

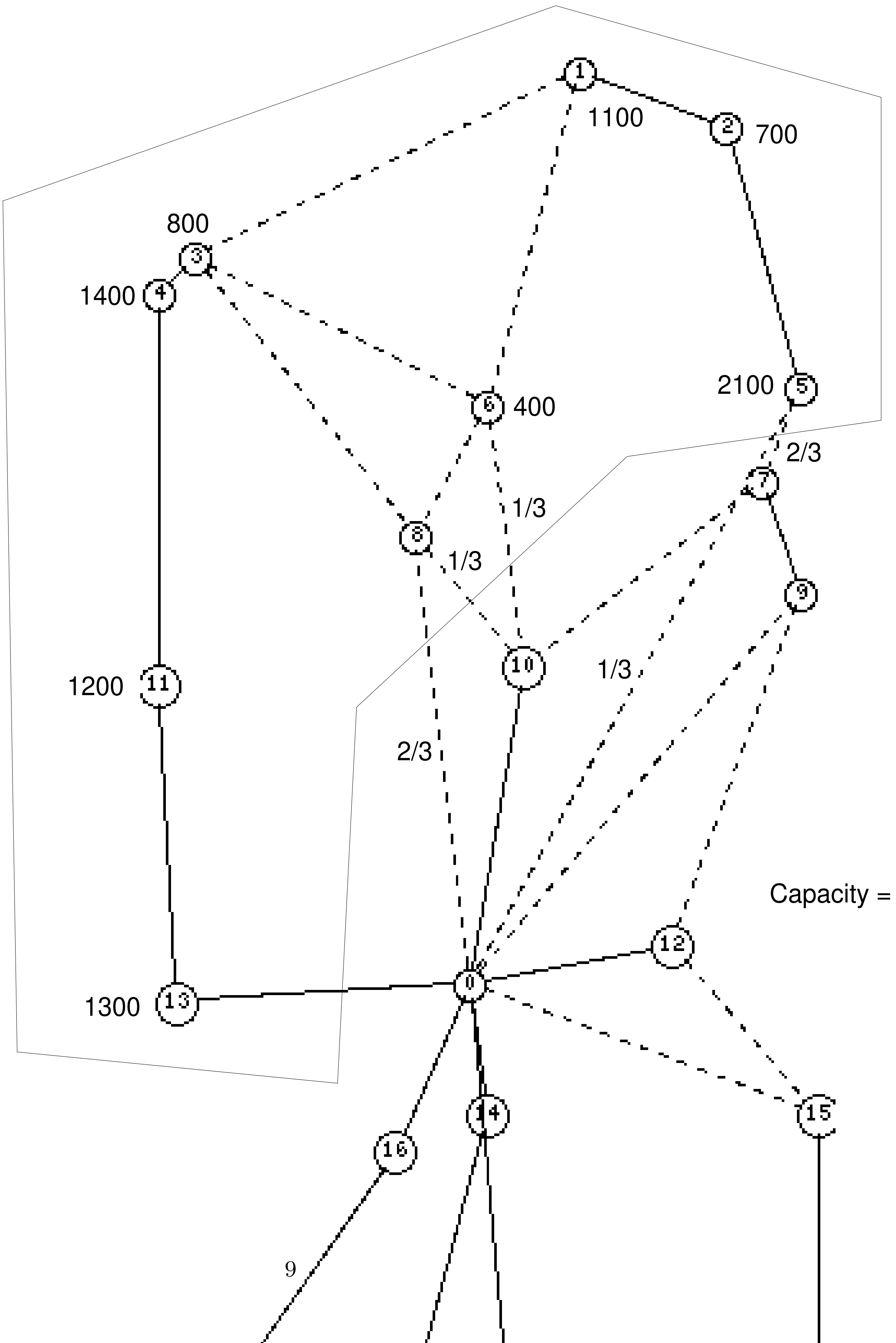
$$\max\{\mathbf{0}^\top \lambda : T\lambda = \hat{x},\ \mathbf{1}^\top \lambda = 1,\ \lambda \geq 0\}$$

  where T is a matrix whose columns are the extreme points of the polytope CPP. This is related to Dantzig-Wolfe decomposition.

- If successful, then separate over the extreme points in the linear combination to obtain a set of possibly violated inequalities. Otherwise, derive an inequality which separates $\hat{x}$ from CPP.


1100
700
800
1400
2100
400
2/3
1/3
1/3
1/3
1200
2/3
Capacity = 6000
1300




1
2
1100
700
800
3
1400
4
2100
5
6
400
2/3
7
1/3
8
1/3
9
10
1/3
1200
11
2/3
Capacity =
12
1300
13
0
14
15
16

The Key Results

Theorem 1 *Let $\hat{x} = T\hat{\lambda}$ be the solution of some decomposition LP. Then if every column t of T with $\hat{\lambda}_t > 0$ satisfies the constraints in $\mathcal{S}$, then so must $\hat{x}$.*

Corollary 1 *Let* $\mathcal{V} = \{(a, \beta) \in \mathcal{S} : a\hat{t} < \beta \text{ for some } \hat{t} \text{ such that } \hat{\lambda}_{\hat{t}} > 0\}$. *Then* $\{(a, \beta) \in \mathcal{S} : a\hat{x} < \beta\} \subseteq \mathcal{V}$.

Theorem 2 *Given $\hat{x} \in \mathbf{R}^n$, we have the following mutually exclusive alternatives:*

> **(I)** *There exists* $p \in \mathbf{Z}_+,\ 2 \le p \le |E|;\ \lambda_1, \ldots, \lambda_p,\ \sum_{i=1}^{p} \lambda_i = 1,\ \lambda_i > 0 \text{ for } i = 1, \ldots, p;$ *and* $S_1, \ldots, S_p,\ S_i \in \mathcal{F} \text{ for } i = 1, \ldots, p$ *such that* $\sum_{i=1}^{p} \lambda_i x^{S_i} = \hat{x}$.
>
> **(II)** *There exists $y \in \mathbf{R}^E$ and $\gamma \in \mathbf{R}$ such that (y, γ) is a valid inequality for* CPP *but $y\hat{x} < \gamma$.*

The Basic Algorithm – Details

In theory, we can solve the decomposition LP by column generation.

Step 1 Generate a matrix T' containing a small subset of columns from T.

Step 2 Solve the Decomposition LP with T' using the dual simplex algorithm. If this LP is feasible, then we have a decomposition.

Step 3 Otherwise, let r be the row in which the dual unboundedness condition was discovered, and let $\mu \in \mathbf{R}^{n+1}$ be the r^{th} row of B^{-1}.

Step 4 Solve CP with cost vector c, consisting of the first n components of μ. Let t be the incidence vector of the result. If $ct < -\mu_{n+1}$, then t is a column eligible to enter the basis. Add t to T' and go to Step 1.

Step 5 Otherwise, $(c, -\mu_{n+1})$ is a valid inequality for CPP which is violated by $\hat{x}$.

Extensions to the Basic Algorithm

- The main drawback of the basic algorithm is the difficulty of generating columns of the matrix T.
- Possible solution: If the fractional graph is sparse, generate columns using an enumerative search.
- The enumeration can optionally be limited to columns that
  - "conform" to the current fractional solution,
  - are feasible for the constrained problem, and
  - have cost less than the current upper bound.
- Advantages of incomplete enumeration
  - many fewer columns – quicker enumeration.
  - stronger Farkas cuts.
- If no columns are found, then we can impose a *No-columns Cut*.

No-columns Cuts

If no feasible (low cost) columns are found that conform to the current fractional solution, then the following cut can be imposed:

$$\sum_{e \in R} x_e - \sum_{e \in E \setminus E(\hat{x})} x_e \leq |R| - 1 \tag{1}$$

where $R = \{e : \hat{x}_e = 1\}$ and $E(\hat{x})$ is the edge set of the support graph of $\hat{x}$ of the current fractional solution.

<u>Observation:</u>

If we can generate all conforming, feasible (low cost) columns, we can ALWAYS generate a *no-columns cut!* We may also get a new upper bound.

Cut Generation Strategy for the VRP

- The degree constraints are always enforced.
- Separation is done only for the capacity constraints from the IP formulation.
- We use the following separation strategy:
  - If the solution is integral, check for connectedness and capacity violations.
  - Otherwise, find the (2-edge) connected components of the support graph without the depot.
  - Shrink all edges in the support graph with weight greater than one and look for violated constraints.
  - Failing that, apply the greedy shrinking algorithm of Nadeff *et al.*
  - Finally, apply the *Decomposition Algorithm*, if appropriate.

VRP Implementation Details

- We used the COMPSys library to implement a branch and cut algorithm.
- Various branching strategies for cuts and variables were tested.
- For computational efficiency, we implemented decomposition as follows:
  - It was run independently on each connected component of $G(\hat{x}) \setminus \{0\}$.
  - It was only run on components whose density was below a specified threshold.
  - All one-paths were contracted prior to running the algorithm.
  - It was only re-run within a search node if the gap had narrowed by at least a specified amount since the last call.

Solving the VRP Using COMPSys

The Master Process:

- Upper bounding consists of multiple heuristics run in parallel in several phases:
  - clustering,
  - routing, and
  - exchanging.
- Construction of the root node consists of:
  - selection of initial edge set, and
  - preprocessing.

The Tree Manager: Completely generic.

The Cut Pool: Generic except for cut storage data structures.

The LP Solver:

- Logical fixing, and
- Column generation.

Computational Results

	Min Tree Size			Min Run Time			*ANR*	Weak Decomposition		
Problem	Size	Depth	Time	Size	Depth	Time	Time	Size	Depth	Time
eil13	3	2	0.29	3	2	0.16	-	1	0	0.23
eil22	1	0	0.09	1	0	0.10	5	1	0	0.16
eil23	1	0	0.09	1	0	0.08	4	1	0	0.15
eil30	44	9	8.09	117	17	7.33	30	68	10	7.98
eil31	8	4	2.25	7	2	1.02	-	4	2	2.02
eil33	2	1	1.14	2	1	0.84	36	2	1	1.68
bayg29	3	2	0.71	5	3	0.53	-	4	2	0.93
bays29	5	3	1.63	7	3	0.92	-	6	3	1.17
ulysses16	1	0	0.10	1	0	0.12	-	1	0	0.19
ulysses22	1	0	0.12	1	0	0.12	-	1	0	0.22
gr17	1	0	0.08	1	0	0.08	-	1	0	0.16
gr21	1	0	0.23	1	0	0.23	-	1	0	0.33
gr24	5	3	2.12	4	2	0.78	-	4	2	0.90
fri26	5	4	0.68	10	4	0.60	-	14	6	1.17
swiss42	13	4	9.69	24	7	4.78	-	14	5	11.97
att48	156	15	135.41	328	17	58.45	-	370	14	172.88
gr48	15	5	9.52	24	8	6.18	-	22	5	7.17
hk48	50	9	64.93	77	8	31.89	-	75	8	37.03
eil51	18	7	45.20	17	5	14.45	342	17	5	14.69
Total	333		282.37	631		128.65	417	607		261.02

Results with the sequential version of COMPSys on a Pentium II running Solaris with CPLEX.

<u>Columns 1-3:</u> Settings to minimize the number of nodes.

<u>Columns 4-6:</u> Settings to minimize the running time.

<u>Columns 7-9:</u> Same as columns 4-6 with decomposition (sparse graphs and feasible columns only).

	Without Decomposition		With Decomposition	
Problem	Size	Time	Size	Time
eil76k7	121,811	288,640	115,991	278,613
eil76k8	484,245	1,927,422	-	-
eil101	244,968	1,900,671	-	-

Results with the parallel version of COMPSys on an IBM SP2

running AIX. The LP engine is CPLEX.

Further Results for Small Problems

	No Decomposition			Weak Decomposition			Strong Decomposition		
Problem	Size	Depth	Time	Size	Depth	Time	Size	Depth	Time
eil13	3	2	0.16	1	0	0.24	1	0	0.30
eil22	1	0	0.10	1	0	0.17	1	0	0.17
eil23	1	0	0.08	1	0	0.17	1	0	0.16
eil30	117	17	7.33	104	13	181.46	80	12	93.32
eil31	7	2	1.02	1	0	2.83	4	2	26.84
eil33	2	1	0.84	2	1	1.70	2	1	7.01
bayg29	5	3	0.53	3	1	2.64	4	2	54.27
bays29	7	3	1.63	6	3	308.73	5	3	101.79
ulysses16	1	0	0.12	1	0	0.18	1	0	0.18
ulysses22	1	0	0.12	1	0	0.20	1	0	0.20
gr17	1	0	0.08	1	0	0.15	1	0	0.15
gr21	1	0	0.23	1	0	0.31	1	0	.31
gr24	4	2	0.78	4	2	5.62	4	2	9.43
fri26	10	4	0.60	10	5	1.90	11	5	195.26
Total	161		12.05	137		506.30	117		297.05

Running times are for the sequential version of COMPSys on a Pentium II running Solaris.

The LP engine used is CPLEX.

Columns 1-3: Settings to minimize the running time.

Columns 4-6: Same as columns 1-3 with decomposition (dense graphs and feasible columns only).

Columns 7-9: Same as columns 1-3 with decomposition (dense graphs and infeasible columns allowed).

More Results for Small Problems

	Weak Cut Generation			Strong Decomposition		
Problem	Size	Depth	Time	Size	Depth	Time
eil13	2	1	0.13	1	0	0.18
eil22	1	0	0.09	1	0	0.18
eil23	1	0	0.07	1	0	0.16
eil30	241	15	13.37	124	15	41.78
eil31	28	8	1.08	17	7	2.92
eil33	61	9	8.04	33	10	187.28
bayg29	18	7	1.06	15	5	226.85
bays29	54	10	4.35	9	4	115.04
ulysses16	1	0	0.12	1	0	0.22
ulysses22	4	2	0.23	5	3	0.59
gr17	1	0	0.08	1	0	0.17
gr21	4	2	0.31	1	0	0.42
gr24	22	6	1.79	15	5	37.05
fri26	8	5	0.38	11	6	1.32
Total	446		31.11	235		614.16

Running times are for the sequential version of COMPSys on a

Pentium II running Solaris. The LP engine used is CPLEX.

<u>Columns 1-3:</u> Weaker cut generator.

<u>Columns 4-6:</u> Same as columns 1-3 with decomposition (dense graphs and infeasible columns allowed).

Conclusions

- No-columns cuts and the Farkas cuts are relatively weak in their current form.
- Although the cuts are effective at reducing the size of the tree, they generally do not reduce running times.
- If allowed to generate infeasible columns, the algorithm can find capacity cuts not found by any other algorithm.
- Overall, the greedy shrinking algorithm is the most efficient and effective algorithm for finding capacity cuts.
- Strong branching with 5-10 candidate variables is extremely effective – branching on cuts is not effective.
- Our test set needs updating!!

Future Work

- Lift the Farkas and No-columns cuts to make them stronger.
- Further improve the efficiency of the column enumeration/generation.
- Further improve the criteria by which it is decided whether or not to call decomposition.
- Exploit the information derived from the column enumeration more effectively.
- Use enumeration to improve upper bounds (currently not done in the VRP).
- Implement dynamic column generation for the decomposition LP.
- Separate over more classes of cuts – borrow technology from the TSP.
- Update the test set.
- Find new classes of problems to which the algorithm can be applied.