GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[X] ORIGINAL
[ ] REVISION NO. ____

Project No. E-24-631 DATE: 4/3/81

Project Director: Dr. R. L. Rardin School/~~Dept~~ ISyE

Sponsor: National Science Foundation; Washington, D. C. 20550

Type Agreement: Grant No. ECS-8018954

Award Period: From 3/15/81 To 8/31/83* (Performance) ---- (Reports)

Sponsor Amount: $49,951 Contracted through:

Cost Sharing: $6,399 (E-24-347) GTRI/~~GTRC~~

Title: Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks

ADMINISTRATIVE DATA OCA CONTACT Duane Hutchison x 4820

1) Sponsor Technical Contact: A. H. Haddad, NSF Program Officer; System Theory and Operation Research; Division of Electrical, Computer, and Systems Engineering; Directorate for Engineering and Applied Science; NSF; Washington, D.C. 20550 (202) 357-9618

2) Sponsor Admin./Contractual Contact: W. A. Bryant, NSF Grants Official; Section II; AAEO/EAS Branch; Division of Grants and Contracts; Directorate for Administration; NSF; Washington, D.C. 20550; (202) 357-9602

Reports: See Deliverable Schedule Security Classification: None

Defense Priority Rating: None

RESTRICTIONS

See Attached NSF Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval - Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with GIT.

COMMENTS: * Includes the usual six (6) month unfunded flexibility period.

COPIES TO:

Administrative Coordinator
Research Property Management
Accounting Office
Procurement Office
Research Security Services
Reports Coordinator (OCA)
Legal Services (OCA)
Library, Technical Reports
EES Information Office (2)
Project File (OCA)
Other: ____

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date February 13, 1984

Project No. E-24-631 School/~~XXX~~ ISyE

Includes Subproject No.(s) -----

Project Director(s) Dr. R. L. Rardin GTRI / ~~GIT~~

Sponsor National Science Foundation; Washington, D.C. 20550

Title Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks

Effective Completion Date: 8/31/83 (Performance) 11/30/83 * (Reports)

* no charges allowed past 8/31/83 even though 90 days are allowed for submissive of final report.

Grant/Contract Closeout Actions Remaining:

- [ ] None
- [x] Final ~~Invoice or Final Fiscal Report~~ FCTR
- [ ] Closing Documents
- [x] Final Report of Inventions
- [ ] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other _____

Continues Project No. _____ Continued by Project No. _____

COPIES TO:

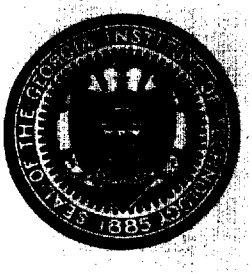
Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/EES Supply Services
~~Research Security Services~~
~~Reports Coordinator (OCA)~~
Legal Services

Library
GTRI
Research Communications (2)
Project File
Other _____

Form OCA 60:1028

E-24-631

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING



GEORGIA INSTITUTE
OF TECHNOLOGY
ATLANTA, GEORGIA 30332

June 1982

PROGRESS REPORT:

Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks

(Grant No. ECS-8018954)

by

Ronald L. Rardin, Ph.D
Principal Investigator

and

R. Gary Parker, Ph.D
Co-Principal Investigator

for

Dr. Abe Haddad
Electrical, Computer and Systems Engineering
National Science Foundation

1. Background

A vast number of important integer and combinatorial problems in areas such as distribution, communications, transportation, and facilities location can be viewed as fixed charge flow problems on graphs or networks, i.e. flow problems with fixed costs incurred on arcs with positive flow. On May 1, 1980 Drs. Ronald L. Rardin and R. Gary Parker proposed to the National Science Foundation a line of research on such fixed charge problems entitled, "Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks." Their proposal envisioned a program of research on forming, and implementing in algorithms, non-standard relaxations of fixed charge problems on networks and graphs. More specifically, the proposal contemplated study of both tight linear programming relaxations of fixed charge problems on graphs and networks, and investigation of combinatorial relaxations for the same problems. Total funding sought was $152,080.

At the request of the National Science Foundation, a revised proposal was submitted on January 13, 1981 for a reduced scope effort considering only the linear programming relaxations. That revised scope was funded as Grant Number ECS-8018954 for two years beginning March, 1981, in the amount of $49,951. This report briefly summarizes progress on the planned research during the first grant year and activities contemplated for the second.

2. Progress during the First Grant Year

As noted above research planned under the grant centers on tight non-standard linear programming relaxations for fixed charge problems on graphs and networks. The relaxations are tight in the sense that

solutions obtained from such relaxations closely approximate optimal solutions for the underlying mixed-integer programming problems. The proposed method of approach anticipated a combined computational and theoretical investigation of such relaxations, with computational phases seeking effective strategies for dealing with the massive linear programs involved in such relaxations and theoretical studies aimed at sharpening the relaxations and proving their effectiveness on restricted classes of problems.

The attached working papers detail how substantial progress has been achieved on both these fronts during the first grant year. Attachment 1

"Tight Relaxations of Fixed Charge Network Flow Problems"

merely summarizes work prior to the beginning of the grant. It has been submitted for publication in Operations Research Letters. Attachment 2,

"Development of a Progressive Disaggregation Approach to Fixed Charge Network Flow Problems"

centers on new computational aspects. Attachment 3,

"Some Polynomially Solvable Multi-Commodity Fixed Charge Network Flow Problems"

includes new theoretical developments. The latter paper has been submitted for publication in Discrete Applied Mathematics.

Briefly, the achievements reported in the papers are the following:

- A variant on our earlier formulation has been discovered which leads to both a tighter linear programming formulation of the problem and (generally) fewer linear programming constraints. Attachment 2, Section 2 provides details.
- Central issues have been isolated, and algorithmic strategies for dealing with them proposed, to implement the disaggregation

concept on our linear programming relaxations progressively. The relaxations involve disaggregation of flows into components tracking the supply point at which the flow began and the demand point to which it is destined. Rather than dealing with all such variables, and associated constraints, at one time, the progressive approach treats flows in increasingly more detailed supply and demand groupings. Ultimately, a full disaggregation to individual supplies and demands may be reached, but it is hoped that the progressive strategy will lead to less total computations by diminishing the effort expanded on early iterations. Among the issues dealt with in Attachment 2 are the form supply and demand groups should take, and how the progressive approach can be integrated in Lagrangean relaxation of the tight form.

- Our relaxations have been proved exact on a significant class of problems arising on graphs with specific structure. Attachment 3 details a proof that the linear programming relaxation we have been studying yields an integer solution for uncapacitated problems on series-parallel graphs--an important subset of planar graphs. The tight linear programming relaxation for such fixed charge network problems is unimodular, regardless of the number of commodities considered. Thus, polynomial procedures for linear programming yield polynomial time algorithms for all such fixed charge cases.

We noted above that two of the attached papers have already been submitted for publication. It is also anticipated that work on

computational phases will be published, but submission awaits computational testing of concepts developed in the research.

Beyond these efforts to disseminate results through scholarly journals, six seminars have been presented by investigators on the work:

- "Progressive Relaxation of Fixed Charge Network Flow Problems," presented to the Fall Joint National meeting of the Operations Research Society of America and The Institute of Management Sciences, Toronto, Canada, April, 1981.
- "Tight Relaxations of Fixed Charge Network Problems", presented in seminars at
    a. Department of Industrial Engineering, Auburn University, April, 1981
    b. Department of Industrial Engineering, State University of New York at Buffalo, April, 1981
    c. Department of Industrial Engineering and Operations Research, Virginia Polytechnic and State University, February, 1982
    d. School of Industrial Engineering, Purdue University, March, 1982
- "Lagrangean Relaxation with Application to Fixed Charge Network Flows", presented to the Applied Mathematics Round Table, Exxon Corporation, March, 1982.

3. Anticipated Activities for the Second Grant Year

Planned activities for the second grant year will be directed toward

completing and extending the achievements described above. More specifically,

- Computational Testing. Algorithmic strategies for the progressive strategy presented in Attachment 2 are being implemented in a computer code so that they can be empircally tested. This computational activity has proceeded more slowly than originally planned because of inadequate availability of computer resources at Georgia Tech. However, satisfactory arrangments have now been made and testing is advancing. By the end of the project both a code and an empirical evaluation of the progressive strategy should be available.
- Polynomial Algorithm. Attachment 3 shows that a significant class of fixed charge problems can be solved exactly via our linear programming relaxation. By appeal to the availability of polynomially-bounded algorithms for linear programming, that result proves the indicated problems are polynomially solvable. However, we believe there should be more direct combinatorial algorithms for the relaxation in such cases. Theoretical effort in the remaining part of the project will be directed toward the discovery of such algorithms.

ATTACHMENT 1

Industrial Systems and Engineering
Report Series No. J-82-3
January, 1982

TIGHT RELAXATIONS OF FIXED CHARGE NETWORK FLOW PROBLEMS

by

Ronald L. Rardin*

* Associate Professor, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract: A vast number of important engineering and management problems can be viewed as network flow problems with fixed charges for opening arcs. This research derives new, tight, linear programming relaxations for such problems based on a disaggregation of flows. The concenpt behind such relaxations is presented, and an algorithm for their solution is discussed.

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS-801954.

1. Introduction

A vast number of important engineering and management problems in distribution, communication, transportation, and facilities location can be viewed as single or multi-commodity network flow problems with fixed charges for constructing/setting up/installing arcs. Such problems with commodities in P can be stated in mixed-integer form as follows:

$$\min \quad \sum_{p \in P} v^p x^p + fy \tag{1}$$

(MFP) s.t.

$$Ex^p = b^p \qquad \text{for all } p \in P \tag{2}$$

$$x^p \geq 0 \qquad \text{for all } p \in P \tag{3}$$

$$(1/u_j) \sum_{p \in P} x_j^p \leq y_j \qquad \text{for all } j \in A \tag{4}$$

$$0 \leq y \leq 1 \tag{5}$$

$$y \text{ integer} \tag{6}$$

Here E is the vertex-arc incidence matrix of a directed graph, $G(V,A)$, x^p is the flow of commodity p on that network, v^p is the variable (per unit) cost of such flow, b^p is a requirements vector for commodity p (having components summing to zero), u_j is the capacity of arc j of A, f_j is the fixed charge on arc j, and y_j is a 0-1 variable switching "on" the fixed charge when flow through arc j is allowed. I assume throughout that all f_j are nonnegative. If capacities, u_j, are not naturally apparent in the problem setting, they can usuallybe generated as any number greater than or equal to the maximum flow through the arc.

Figure 1 shows a simple numerical example with $|P| = 1$ commodity. All 10 units of flow originate at vertex 1; 5 are required at vertex 3 and 5 at vertex 4. It is easy to check that an optimal solution sends one unit 1-4, 4 units

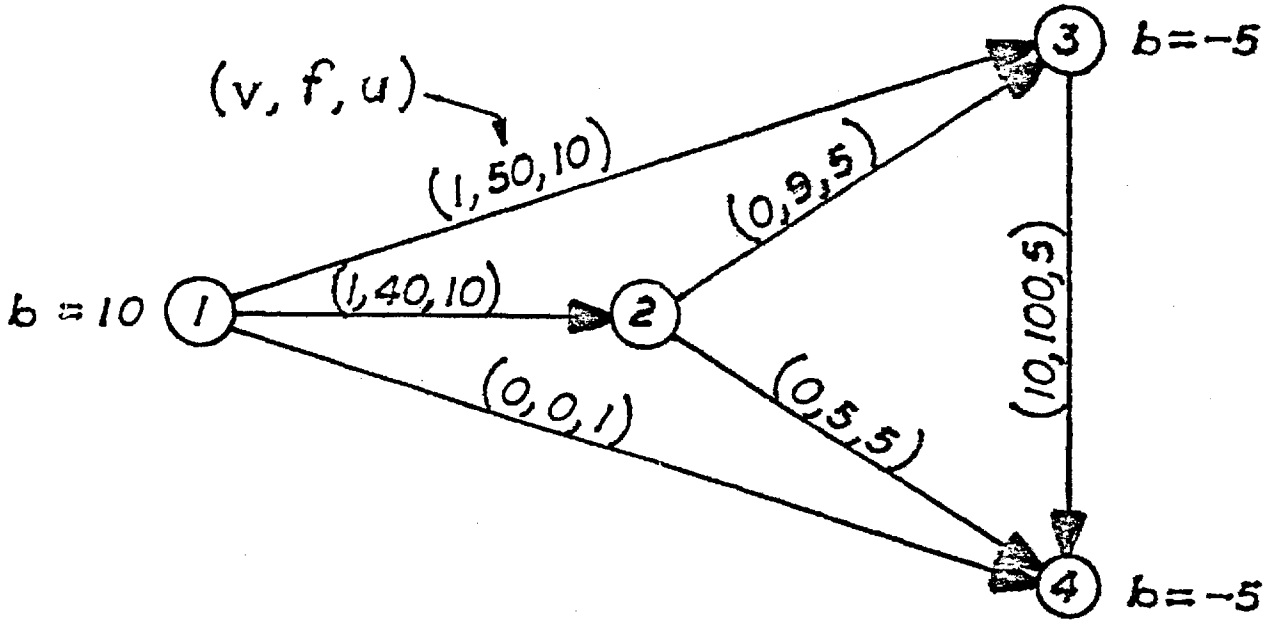


Figure 1. An Example Network

1-2-4, and 5 units 1-2-3. Total cost is 63.

2. The Standard Relaxation

Since the early work of Balinski [1, 2] a standard approach to dealing with problems (MFP) has been to solve linear programming relaxations ($\overline{\text{MFP}}$) obtained when constraints (6) are dropped. Such relaxations provide both bounds for branch-and-bound schemes and a source of approximate solutions; given an ($\overline{\text{MFP}}$) optimum, one need only round "up" all positive, but fractional y_j to obtain a feasible solution to (MFP).

For the above example this ($\overline{\text{MFP}}$) relaxation is solved by sending 1 unit 1-4, 4 units 1-2-4, and 5 units 1-3; total cost is 54 (83% of the optimal 63). When rounded "up" this solution costs 104 (165% of optimal).

Neither of these values is very satisfactory, and actual experience is often much worse. To see why, observe that the constraints (4) will always be tight in some optimal solution for ($\overline{\text{MFP}}$); where $f_j > 0$, slack in such constraints implies unnecessary cost. Since constraints (4) will be equalities in ($\overline{\text{MFP}}$), their effect is to prorate the fixed cost, f_j, over the corresponding capacity. For example, in arc (1-2) of Figure 1, 4/10 of the fixed cost, 40, would be paid in the ($\overline{\text{MFP}}$) optimum because 4/10 of the capacity, 10, is used by the optimal flow. If capacities are large, it is easy to see that this proration process would soon negate, or nearly negate, the impact of fixed costs on ($\overline{\text{MFP}}$) optima. Optimal relaxation solutions tend to use many arcs at relatively small fractions of capacity. This is particularly so when capacities are artificially created as maximum possible flows.

3. A Disaggregated Formulation

In a number of special cases, including warehouse location problems (Balinski [2], Davis and Ray [4], Erlenkotter [5], Bilde and Krarup [3], Geoffrion and Graves [7]) and uncapacitated problems (Magnanti and Wong [8]), various researchers have

shown the merit of disaggregating (MFP) flows to obtain linear programming relaxations that more closely approximate the mixed-integer problems. One can retrieve these special cases and extend the notion to all (MFP) by recognizing that flow in any commodity can always be disaggregated into separate commodity flows between origin-destination pairs of the requirements vector, b^p. Specifically, let $x^p[s,t]$ be a vector showing the flow of the portion of commodity p originating at source s and destined for sink t. Then an equivalent mixed-integer form to (MFP) is

$$\min \sum_{p \in P} \left(\sum_{s \in S_p} \sum_{t \in T_p} x^p[s,t] \right) + f\, w \tag{7}$$

$$\text{s.t. } E^p x^p[s,t] = 0 \quad \text{for all } p \in P,\ s \in S_p,\ t \in T_p \tag{8}$$

$$\sum_{t \in T_p} \sum_{\{j \text{ leaving } s\}} x_j[s,t] = b^p_s \quad \text{for all } p \in P,\ s \in S_p \tag{9}$$

$$-\sum_{s \in S_p} \sum_{\{j \text{ entering } t\}} x_j[s,t] = b^p_t \quad \text{for all } p \in P,\ t \in T_p \tag{10}$$

$$\text{(DFP)} \qquad x^p[s,t] \geq 0 \quad \text{for all } p \in P,\ s \in S_p,\ t \in T_p \tag{11}$$

$$(1/u_j) \sum_{p \in P} \sum_{s \in S_p} \sum_{t \in T_p} x^p_j[s,t] \leq w_j \quad \text{for all } j \in A \tag{12}$$

$$(1/-b^p_t) \sum_{s \in S_p} x^p_j[s,t] \leq w_j \quad \text{for all } j \in A,\ p \in P,\ t \in T_p \tag{13}$$

$$(1/b_s^p) \sum_{t \in T_p} x_j^p[s,t] \leq w_j \quad \text{for all } j \in A,\ p \in P,\ s \in S_p \tag{14}$$

$$0 \leq w \leq 1 \tag{15}$$

$$w \text{ integer} \tag{16}$$

Here S_p = {sources for commodity p} = $\{s: b_s^p > 0\}$

T_p = {sinks for commodity p} = $\{t: -b_t^p > 0\}$

E^p = the row submatrix of E containing row $i \in \{i: b_i^p = 0\}$

In this new form w corresponds directly to y of (MFP), and flow variables are related by

$$x_j^p = \sum_{s \in S_p} \sum_{t \in T_p} x_j^p[s,t] \tag{17}$$

Relaxations (7), (8) through (10), (11), (12), (15), and (16) of (DFP) correspond to (1), (2), (3), (4), (5), and (6) of (MFP), respectively. Denote by $\nu(\cdot)$ the value of an optimal solution to problem $(\cdot)$ and by $(\overline{DFP})$ the linear programming relaxation of (DFP). Then this correspondence and the fact that (DFP) and $(\overline{DFP})$ have extra constraints (13) and (14) lead to the following conclusion:

Proposition 1. Solution values for (MFP), (DFP), $(\overline{MFP})$ and $(\overline{DFP})$ satisfy

$$\nu(\overline{MFP}) \leq \nu(\overline{DFP}) \leq \nu(DFP) = \nu(MFP) \tag{18}$$

The new elements in the (DFP) formulation are systems (13) and (14). Intuitively, (13) requires that , w_j, the portion of the fixed charge paid on arc j, must equal or exceed the fraction of a demand satisfied through arc j.

Similarly, (14) forces w_j to match the portions of each supply directed through arc j. The extra constraints are implied by (12) when integrality, (16), is enforced. But they may significantly improve the linear programming relaxation $(\overline{DFP})$ because f_j is now prorated over both u_j and all relevant supplies and demands. The latter are often much smaller than capacities.

The example of Figure 1 illustrates. An optimal solution to the linear programming relaxation $(\overline{DFP})$ sends 1 unit 1-4, 4 units 1-2-4, and 5 units 1-2-3. The relaxation cost is 62, 98% of the optimal 63. When all fractional w_j in the relaxation are rounded "up", a feasible solution is obtained that is indeed the (DFP) optimum. The effect of the disaggregation is seen on arc (1,2). The $(\overline{DFP})$ optimum pays the entire fixed charge, 40, because all demand at vertex 3 is satisfied through (1,2). From this example we may draw the further conclusion:

Proposition 2: In selected problems both inequalities of (18) may be strict.

4. Solving the Tighter Relaxation

If the strength of the $(\overline{DFP})$ relaxation is to be realized, an approach must be found for solving or nearly solving that massive linear program. Three cases can be identified. Uncapacitated cases have neither binding arc capacities, u_j, nor limits on supply at sources. Equivalently they are problems where constraints (12) are unnecessary and each requirements vector has only one positive component at the commodity's single source. Weakly capacitated cases admit supply limits, but do not have binding arc capacities. They include the capacitated warehouse location problem. Finally, fully capacitated problems have binding arc capacities, and possibly also binding supplies.

In both the uncapacitated and the weakly capacitated cases we can ignore constraints (12) of (DFP). Suppose we "dualize" (13) and (14), i.e. place them

in the objective function with nonnegative dual multipliers $\delta_j^p[t]$ and $\sigma_j^p[s]$, respectively, to obtain

$$\min \sum_{p\in P} v^p \left(\sum_{s\in S_p} \sum_{t\in T_p} x^p[s,t] \right) + fw$$

$$(DFP_{\delta\sigma}) \quad + \sum_{p\in P} \sum_{j\in A} \sum_{t\in T_p} \delta_j^p[t] \left(\frac{1}{-b_t^p} \sum_{s\in S_p} x_j^p[s,t] - w_j \right) \tag{19}$$

$$+ \sum_{p\in P} \sum_{j\in A} \sum_{s\in S_p} \sigma_j^p[t] \left(\frac{1}{b_s^p} \sum_{t\in T_p} x_j^p[s,t] - w_j \right)$$

s.t. (8), (9), (10), (11), (15) and (16)

For fixed δ and σ variables in $(DFP_{\delta\sigma})$ the commodities are linked only at sources and sinks (through (9) and (10)). Moreover, each origin-destination commodity problem is essentially one of picking a single path along which to ship from source to sink. Thus, one can approach $(\overline{DFP})$ by trying to maximize $v(DFP_{\delta\sigma})$ over nonnegative values of the dual variables as follows:

Step 0: Initialization. Set all $\delta_j^t[t]$ and $\sigma_j^p[s]$ to zero, and fix dual and primal incumbent solution values $v_D^* \leftarrow -\infty$, $v_p^* \leftarrow +\infty$.

Step 1: Implicit Costs. Determine (19) objective function coefficients

$$\tilde{f}_j \leftarrow f_j - \sum_{p\in P} \sum_{s\in S_p} \sigma_j^p[s] - \sum_{p\in P} \sum_{t\in T_p} \delta_j^p[t] \tag{20}$$

$$\tilde{v}_j^p[s,t] \leftarrow v_j^p + \sigma_j^p[s]/b_s^p + \delta_j^p[t]/(-b_t^p) \tag{21}$$

Step 2: Shortest Paths. For each $p\in P$, $s\in S_p$, $t\in T_p$ compute the shortest path from s to t over arc lengths $\tilde{v}_j^p[s,t]$. Let $R^p[s,t]$ be the set of arcs in that path and $c^p[s,t]$ its length.

Step 3: Transportation Problems: For each commodity $p \in P$, solve a transportation problem from sources $s \in S_p$ to sinks $t \in T_p$ with costs $c^p[s,t]$. Supplies are $\{b_s^p > 0\}$ and demands $\{-b_t^p > 0\}$. Denote by $z^p[s,t]$ an optimal flow from s to t in that transportation problem.

Step 4: Flow Solution: Construct and optimal flow for $(DFP_{\delta\sigma})$ by assigning for each p, $s \in S_p$, $t \in T_p$, $z^p[s,t]$ units of flow along all arcs in the corresponding shortest path $R^p[s,t]$.

Step 5: 0-1 Problem. Compute relaxation optimal values for the w_j variables via

$$\tilde{w}_j \leftarrow 1 \text{ if } \tilde{f}_j \leq 0 \text{ and } 0 \text{ otherwise.}$$

Step 6: Dual Solution. Compute a dual solution, ν_D, as the sum of the $(DFP_{\delta\sigma})$ costs of the optima in Steps 4 and 5. If $\nu_D > \nu_D^*$, save a new dual incumbent $\nu_D^* \leftarrow \nu_D$.

Step 7: Primal Solution. Create a feasible solution to (DFP) by paying full fixed charges on any arc used in the flow of Step 4. Let ν_p be its cost, and if $\nu_p < \nu_P^*$, save a new primal incumbent $\nu_P^* \leftarrow \nu_p$.

Step 8: Dual Update. If ν_P^* is sufficiently close to ν_D^*, stop and accept the primal incumbent as an approximate (DFP) optimum. If not, modify duals $\delta_j^k[t]$ and $\sigma_j^t[s]$ by taking a finite step along a subgradient of the function $\nu(DFP_{\delta\sigma})$ at the current dual point. Then return to Step 1.

Since every problem $(DFP_{\delta\sigma})$ is a Lagrangean relaxation of (DFP) (see Fisher [6] for details of such relaxations and subgradient search), and every flow of Step 4 is primal feasible we have:

Proposition 3: At any stage of the above algorithm

$$\nu_D^* \leq \nu(DFP) \leq \nu_P^*. \tag{22}$$

5. Preliminary Computational Experience

To see whether values in (22) could be brought close enough together to solve problems without the need for branch and bound, 15 random test problems were generated and approximately solved by the above algorithm. The problems were uncapacitated, 1-true-commodity cases with relatively high fixed charges on all arcs.

Table 1 summarizes problem characteristics and results obtained for the three problems of each size group. As indicated, the ordinary ($\overline{\text{MFP}}$) relaxations provide very poor information. Relaxation solution values are only 25-50% of optima.

The above (DFP) algorithm was set to stop when either $\nu_P^*/\nu_D^* \leq 102.5\%$ or a 15 minute time limit (CDC Cyber 74) was reached. All problems of less than 1000 arcs stopped before time limit. As indicated, the 1000 arc cases reached solutions provably within 4-8% of optimal in the 15 minutes.

Although this amount of computer time is not insignificant, and results are highly preliminary, values in Table 1 strongly suggests that disaggregated relaxation approaches to fixed charge network problems have great promise. Existing branch-and-bound algorithms for such problems (e.g. Rardin and Unger [9]) are taxed at 100-200 fixed charge arcs because of poor ($\overline{\text{MFP}}$) bounds. With (DFP) it appears 1,000 or more arc problems are within range.

Table 1. Preliminary Computational Results

Problem Size			Estimated % $\nu(\overline{MFP})$	CDC Seconds to Reduce ν_P^*/ν_D^*
Arcs	Nodes	Demands	Forms of ν(MFP)	$\leq$ 102.5% with (DFP)
50	20	5	43.5%	0.8
			23.2%	0.8
			54.6%	5.3
100	36	10	47.3%	7.5
			37.1%	3.8
			36.9%	2.7
200	67	20	36.1%	23.5
			37.0%	19.2
			41.3%	19.6
500	157	50	35.9%	416.5
			40.1%	353.2
			47.6%	237.6
1000	308	100	37.9%	105.5% in 900
			29.3%	107.7% in 900
			41.0%	103.8% in 900

REFERENCES

1. M.L. Balinski, "Fixed Cost Transportation Problems," *NRLQ*, 8, 41-54, (1961).

2. M.L. Balinski, "Integer Programming: Method's, Uses , Computation," *Management Science*, 12, 253-313, (1965).

3. O. Bilde and J. Krarup, "Sharp Lower Bounds and Efficient Algorithms for the Simple Plant Location Problem," *Annals of Discrete Mathematics* 1, (1977). (Based on a 1967 technical report in Danish).

4. P. S. Davis and T. L. Ray, "A Branch-Bound Algorithm for the Capacitated Facilities Location Problem," *NRLQ*, 16, 331-344, (1969).

5. D. Erlenkotter, "A Dual-Based Procedure for Uncapacitated Facility Location," working paper No. 261, Western Management Science Institute, University of California, Los Angeles, (July 1977).

6. Marshall L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, 27, 1-18, (1981).

7. A.M. Geoffrion and G.W. Graves, "Multicommodity Distribution System Design by Benders Decomposition," *Management Science*, 20, 822-844, (1974).

8. T. L. Magnanti and R. T. Wong, "Accelerating Benders Decomposition: Enhancement and Model Selection Criteria," *Operations Research*, 29, 464-484, (1981).

9. R. L. Rardin and V. E. Unger, "Solving Fixed Charge Network Problems with Group Theory Based Penalties," *Naval Research Logistics Quarterly*, 23, 67-84, (1976).

ATTACHMENT 2

Industrial and Systems Engineering
Report Series J-82-4
June, 1982

DEVELOPMENT OF A PROGRESSIVE DISAGGREGATION ALGORITHM FOR FIXED CHARGE NETWORK FLOW PROBLEMS

by

Ronald L. Rardin*

and

Oscar Adaniya**

* Associate Professor, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

** Assistant Professor, Industrial Engineering, University of Miami, Box 248294, Coral Gables, Florida 33124

This paper describes preliminary research still in progress. Do not reference or quote without the expressed consent of the authors.

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS-801954

Abstract

Fixed charge network flow problems model network design and location settings by allowing both fixed and variable charges for arc flow. Recent research has shown that very close approximations to mixed-integer solutions for each problems can be obtained from massive linear programs wherein flows are artificially disaggregated into separate components for each origin - destination pair. This paper develops the strategy of a progressive disaggregation algorithm employing the latter linear programming relaxation. However, flows are initially undisaggregated. As computation proceeds, supply and demand subsets are further and further partitioned to tighten the relaxation as required without incurring the computational burden of a complete disaggregation into supply-demand pairs.

1. Introduction

The fixed charge network flow problem in one commodity is typically formulated

$$\min \sum_{(i,j)\in E} v_{ij}x_{ij} + \sum_{(i,j)\in E} f_{ij}y_{ij} \tag{1}$$

$$\text{s.t.} \sum_{(i,\beta)\in E} x_{i\beta} = d_{\beta} \quad \text{for all } \beta\in D \tag{2}$$

$$\sum_{(\alpha,j)\in E} x_{\alpha j} \le s_{\alpha} \quad \text{for all } \alpha\in S \tag{3}$$

(FC)

$$\sum_{(\ell,j)\in E} x_{\ell j} - \sum_{(i,\ell)\in E} x_{i\ell} = 0 \quad \text{for all } \ell\in T \tag{4}$$

$$x_{ij}/u_{ij} \le y_{ij} \quad \text{for all } (i,j)\in E \tag{5}$$

$$x_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{6}$$

$$1 \ge y_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{7}$$

$$y_{ij} \text{ integer} \quad \text{for all } (i,j)\in E \tag{8}$$

Here E is the arc set of a specified network; x_{ij} is the flow from i to j; S, D and T are the supply point, demand point and transshipment point subsets of nodes respectively; s_{α} is the supply at point α; d_{β} is the demand at point β; and u_{ij} is a capacity of arc (i,j) flow. Costs (1) include a variable (per unit flow) cost v_{ij} and a fixed charge f_{ij} "switched on" by the 0-1 variable y_{ij} whenever $x_{ij} > 0$. We assume throughout that all f_{ij} and v_{ij} are nonnegative although the latter requirement can be relaxed in some cases.

Formulation (FC) gives a correct mixed-integer statement of the fixed charge network flow problem, but its linear programming relaxation, (obtained by deleting

constraint (8)) often provides only a very poor approximation to the mixed integer form. Rardin and Choe (1979) and Rardin (1982) demonstrated that a much better linear programming approximation is obtained by disaggregating flows x_{ij} into components $x_{ij}[\alpha,\beta]$ distinguished by the supply point α at which the flow originated and the demand point β to which it is defined.

Such a multi-commodity formulation is

$$\min \quad \sum_{(i,j)\in E} v_{ij} \sum_{\alpha\in S} \sum_{\beta\in D} x_{ij}[\alpha,\beta] + \sum_{(i,j)\in E} f_{ij}\, y_{ij} \tag{9}$$

$$\text{s.t.} \quad \sum_{\alpha\in S} \sum_{(i,\beta)\in E} x_{i\beta}[\alpha,\beta] = d_\beta \qquad \text{for all } \beta\in D \tag{10}$$

$$\sum_{\beta\in D} \sum_{(\alpha,j)\in E} x_{\alpha j}[\alpha,\beta] \le s_\alpha \qquad \text{for all } \alpha\in S \tag{11}$$

(MC)

$$\sum_{(\ell,j)\in E} x_{\ell j}[\alpha,\beta] - \sum_{(i,\ell)\in E} x_{i\ell}[\alpha,\beta] = 0 \qquad \text{for all } \alpha\in S,\ \beta\in D,\ \ell\in T \tag{12}$$

$$(1/u_{ij}) \sum_{\alpha\in S} \sum_{\beta\in D} x_{ij}[\alpha,\beta] \le y_{ij} \qquad \text{for all } (i,j)\in E \tag{13}$$

$$x_{ij}[\alpha,\beta] \ge 0 \qquad \text{for all } (i,j)\in E,\ \alpha\in S,\ \beta\in D \tag{14}$$

$$1 \ge y_{ij} \ge 0 \qquad \text{for all } (i,j)\in E \tag{15}$$

$$y_{ij} \text{ integer} \qquad \text{for all } (i,j)\in E \tag{16}$$

$$\frac{x_{ij}[\alpha,\beta]}{\min\{s_\alpha, d_\beta\}} \le y_{ij} \qquad \text{for all } (i,j)\in E,\ \alpha\in S,\ \beta\in D \tag{17}$$

As mixed-integer programs, forms (FC) and (MC) are equivalent. However, disaggregation of (FC) flows x_{ij} into separate commodities $x_{ij}[\alpha,\beta]$ leads to a tighter linear programming relaxation in (MC) because of the new constraints (17). With $f_{ij} \geq 0$ the linear programming relaxation, say ($\overline{FC}$), of (FC) will always have an optimal solution with no slack in (5). Thus, ($\overline{FC}$) solutions incur only the fraction x_{ij}/u_{ij} of the fixed charge f_{ij} that flow x_{ij} forms of its capacity u_{ij}. Equation (13) enforces the same limit in ($\overline{MC}$), the linear programming, relaxation of (MC). However, (17) also forces y_{ij} to be as large as the fraction of any source α or sink β flow passing through (i,j). The improved linear programming relaxation follows when (as is usually the case), s_α and/or d_β are much smaller than u_{ij}.

Although providing generally much tighter linear programming approximations, the ($\overline{MC}$) form is an enormous linear program. For a case with 750 arcs, 25 supplies, 100 demands, and 125 transshipment nodes, (MC) has over 400 thousand main constraints and approximately 2.2 million variables. The dual ascent scheme proposed by Rardin and Choe (1979) exploits problem structure in a Lagrangean relaxation, (we give details below), but a typical iteration still involves shortest path problems for each (α,β) pair, and search over dual variables for all constraints (17). For the problem size just described, there would be 2500 such shortest path problems and approximately 1.9 million searchable dual variables.

However, the formulations (FC) and (MC) may be viewed as endpoints of a disaggregation continuum. Form (FC) treats all flows in a single commodity; (MC) disaggregates flows into artificial commodities for each origin - destination pair. Certainly, there are intermediate possibilities wherein flow is treated in groups, (A_k,B_k) with $A_k \subset S$, $B_k \subset \mathcal{D}$.

In this paper we first sharpen the (MC) formulation and then develop

strategies for an algorithm exploiting a progressive disaggregation of $S \times \mathcal{D}$ flows. The algorithm generally follows the Lagrangean relaxation philosophy of Rardin and Choe (1979), but processing begins with the undisaggregated form (FC), i.e. with one supply group $A_1 = S$ and one demand group $B_1 = \mathcal{D}$. As computation proceeds supply and demand groups are progressively partitioned to create new artificial commodity structures. It is hoped that computational testing now underway will demonstrate such a progressive approach reduces total calculation to obtain a satisfactory approximation to an ($\overline{\text{MC}}$) optimum.

2. An Improved Formulation

Flow in our given network can be conceptualized as the rectangle of Figure 1. Sides reflect supplies and demands respectively. Formulation (FC), which uses only one commodity, views the rectangle of flows on arc (i,j) as a single unit x_{ij}. In (MC), each supply, demand cell of the rectangle is tabulated separately as $x_{ij}[\alpha,\beta]$. At disaggregation levels between these extremes, supplies and demands are grouped in a rectangle (A_k,B_k) collecting all flows from origins $\alpha\varepsilon A_k$ to destinations $\beta\varepsilon B_k$.

The analog of Rardin and Choe's (MC) constraint (17) for such a commodity (A_k,B_k) is

$$\frac{\sum_{\alpha\varepsilon A_k}\sum_{\beta\varepsilon B_k} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha\varepsilon A_k} s_\alpha,\ \sum_{\beta\varepsilon B_k} d_\beta\right\}} \le y_{ij} \quad \begin{array}{l}\text{for all } (i,j)\varepsilon E \\ \text{and all } k\end{array} \tag{18}$$

However, by treating supplies and demands separately we can expand the sums in the numerator and thus sharpen the relaxation.

Lemma 1: Improved Formulation. Let $x_{ij}[\alpha,\beta]$, s_α, d_β, S and $\mathcal{D}$ be as in formulation (MC), A_k a nonempty subset of S and B_k a nonempty subset of $\mathcal{D}$. Then the following constraints are satisfied by every feasible (integer) solution to (FC)

$$\frac{\sum_{\alpha\varepsilon A_k}\sum_{\beta\varepsilon \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha\varepsilon A_k} s_\alpha,\ \sum_{\beta\varepsilon \mathcal{D}} d_\beta\right\}} \le y_{ij} \quad \begin{array}{l}\text{for all } (i,j)\varepsilon E \\ \text{and all } k\end{array} \tag{19}$$

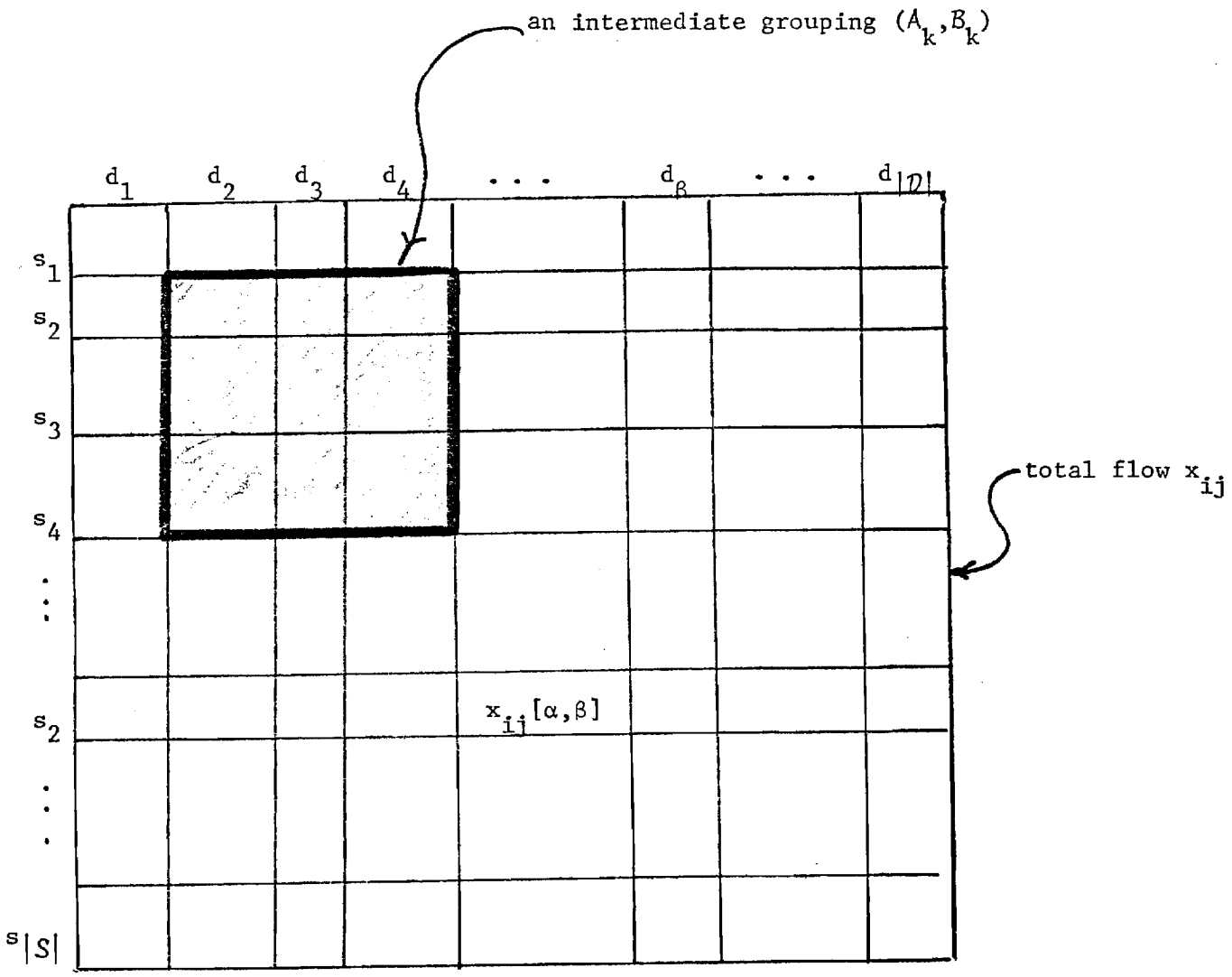


Figure 1: Total Flow as a Supplies by Demands Rectangle

$$\frac{\sum_{\alpha \in S} \sum_{\beta \in B_\ell} x_{ij}[\alpha,\beta]}{\min \left\{ \sum_{\beta \in B_\ell} d_\beta, \sum_{\alpha \in S} s_\alpha \right\}} \leq y_{ij} \quad \text{for all } (i,j) \in E \text{ and all } \ell \quad (20)$$

Furthermore, for specified $s_\alpha > 0$, $d_\beta > 0$, $x_{ij}[\alpha,\beta]$,

$$\{y_{ij} \text{ satisfying (18)}\} \supset \{y_{ij} \text{ satisfying (19) and (20)}\} \quad (21)$$

Proof: It is clear that (19) and (20) are valid in (MC); they simply require that y_{ij} be at least the fraction of supply in A_k or demand in B_ℓ passing through (i,j), respectively. To see (21) observe that if $\sum_{\alpha \in A_k} s_\alpha \leq \sum_{\beta \in B_\ell} d_\beta$, (19) has the same denominator as (18), and at least as great a numerator. If $\sum_{\alpha \in A_k} s_\alpha \geq \sum_{\beta \in B_k} d_\beta$, (20) dominates (18).

■

For a system of q commodities (A_1, B_1), $(A_2, B_2), \ldots, (A_q, B_q)$ there are q constraints of type (18) and potentially 2q like (19) and (20). However, any commodities k and ℓ with $A_k = A_\ell$ or $B_k = B_\ell$ have the same constraint (19) or (20) respectively. The result can be a considerable reduction in the possible number of (19) and (20). In the extreme case where every $(\alpha,\beta) \in S \times D$ forms a separate commodity, there are $|S| + |D|$ constraints (19) and (20), but $|S| \cdot |D|$ limits (18). Thus, at least, as this complete disaggregation is approached, use of (19) and (20) results in both a substantial saving of constraints and a gain in formulation tightness.

3. The Lagrangean Relaxation Setting

With even a partial disaggregation of problem flows into artificial commodities, one obtains a formidable linear program relaxation to be solved. If arc capacities (13) (or (5)) are nonbinding, Rardin and Choe (1979) showed how an effective Lagrangean relaxation of the remaining problem could be structured by summing constraints (19) and (20) in the objective function with nonnegative dual multipliers. Let $A \triangleq \{A_k\}$ be the list of distinct supply subsets of current artificial commodities, $\{\sigma_{ij}[k]: A_k \in A, (i,j) \in E\}$ be the nonnegative dual multipliers on corresponding constraints (19), $\mathcal{B} \triangleq \{B_\ell\}$ the list of distinct demand subsets of current commodities, and $\{\delta_{ij}[\ell]: B_\ell \in \mathcal{B}, (i,j) \in E\}$ be the dual variables on their constraints (20). Then the implied Lagrangean relaxation is as follows:

$$\min \sum_{(i,j)\in E} v_{ij} \sum_{\alpha\in S} \sum_{\beta\in \mathcal{D}} x_{ij}[\alpha,\beta] + \sum_{(i,j)\in E} f_{ij} y_{ij}$$

$$+ \sum_{(i,j)\in E} \sum_{A_k \in A} \sigma_{ij}[k] \left[\frac{\sum_{\alpha\in A_k} \sum_{\beta\in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{ \sum_{\alpha\in A_k} s_\alpha, \sum_{\beta\in \mathcal{D}} d_\beta \right\}} - y_{ij} \right] \tag{22}$$

$$+ \sum_{(i,j)\in E} \sum_{B_\ell \in \mathcal{B}} \delta_{ij}[\ell] \left[\frac{\sum_{\alpha\in S} \sum_{\beta\in B_k} x_{ij}[\alpha,\beta]}{\min\left\{ \sum_{\beta\in B_\ell} d_\beta, \sum_{\alpha\in S} s_\alpha \right\}} - y_{ij} \right]$$

$$\text{s.t.} \sum_{\alpha\in S} \sum_{(i,\beta)\in E} x_{ij}[\alpha,\beta] = d \qquad \text{for all } \beta \in \mathcal{D} \tag{23}$$

$(P_{\sigma\delta}[A,B])$

$$\sum_{\beta\in\mathcal{D}} \sum_{(\alpha,j)\in E} x_{\alpha j}[\alpha,\beta] \le s_\alpha \quad \text{for all } \alpha\in S \tag{24}$$

$$\sum_{(\ell,j)\in E} x_{\ell j}[\alpha,\beta] - \sum_{(i,\ell)\in E} x_{i\ell}[\alpha,\beta] = 0 \quad \text{for all } \alpha\in S,\ \beta\in\mathcal{D},\ \ell\in T \tag{25}$$

$$x_{ij}[\alpha,\beta] \ge 0 \quad \text{for all } (i,j)\in \bar{E},\ \alpha\in S,\ \beta\in\mathcal{D} \tag{26}$$

$$1 \ge y_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{27}$$

$$y_{ij} \text{ integer} \quad \text{for all } (i,j)\in E \tag{28}$$

For any choice of nonnegative $\delta_{ij}[k]$ and $\sigma_{ij}[\ell]$ formulation, $(P_{\sigma\delta}[A,B])$ gives a valid lower bound on the cost of an (FC) or (MC) optimum. A search is, of course, necessary to find good dual values.

The advantage of the $(P_{\sigma\delta}[A,B])$ form lies with the fact that $[\alpha,\beta]$ systems are linked only through the objective function. Thus, for fixed dual values, $(P_{\sigma\delta}[\alpha,\beta])$ separates into a series of shortest path problems for $[\alpha,\beta]$ pairs, followed by an S to $\mathcal{D}$ transportation problem.

Including subgradient steps to improve duals and revise the present commoditization, a full procedure employing $(P_{\sigma\delta}[A,B])$ is as follows:

Step 0: Initialization. Fix dual and primal incumbent values $\nu_D^* \leftarrow -\infty$, $\nu_P^* \leftarrow +\infty$.

Step 1: Initial Disaggregation. Partition the source mode by destination node set $S\times\mathcal{D}$ into an initial series of artificial supply-demand commodities, and let A be the list of distinct supply subsets, A_k,

and $\mathcal{B}$ the corresponding list of distinct demand subsets, $\mathcal{B}_\ell$. Fix all duals $\sigma_{ij}[k]$ and $\delta_{ij}[\ell]$ at zero.

Step 2: Implicit Costs. Determine (22) objective function coefficients

$$\tilde{f}_{ij} \leftarrow f_{ij} - \sum_{A_k \in \mathcal{A}} \sigma_{ij}[k] - \sum_{\mathcal{B}_\ell \in \mathcal{B}} \delta_{ij}[\ell] \tag{29}$$

$$\tilde{v}_{ij}[\alpha,\beta] \leftarrow v_{ij} + \sum_{\{A_k \in \mathcal{A}: \alpha \in A_k\}} (\sigma_{ij}[k]/s[k]) + \sum_{\{\mathcal{B}_\ell \in \mathcal{B}: \beta \in \mathcal{B}_\ell\}} (\delta_{ij}[\ell]/d[\ell]) \tag{30}$$

where
$$s[k] \triangleq \min \left\{ \sum_{\alpha \in A_k} s_\alpha, \sum_{\beta \in \mathcal{D}} d_\beta \right\} \tag{31}$$

$$d[\ell] \triangleq \min \left\{ \sum_{\beta \in \mathcal{B}_\ell} d_\beta, \sum_{\alpha \in \mathcal{S}} s_\alpha \right\} \tag{32}$$

Step 3: Shortest Paths. For each pair (α,β) of a source and a destination node, compute the shortest path from every α to every β over arc lengths $\tilde{v}_{ij}[\alpha,\beta]$. Let $R[\alpha,\beta]$ be the set of arcs in the shortest path from node α to node β and $c[\alpha,\beta]$ its length.

Step 4: Transportation Problem. Using costs $c[\alpha,\beta]$, suppliers s_α and demand d_β, solve an $\mathcal{S}$ to $\mathcal{D}$ transportation problem. Denote by $z[\alpha,\beta]$ an optimal flow from α to β obtained in the solution to the transportation problem.

Step 5: Flow Solution: For each α and β , assign $z[\alpha,\beta]$ units of flow to all arcs (i,j) in the corresponding set of shortest path arcs $R[\alpha,\beta]$.

Step 6: 0-1 Problem. Compute relaxation optimal values for the y_{ij} variables via

$$y_{ij} \leftarrow 1 \quad \text{if } \tilde{f}_{ij} \leq 0 \text{ and } 0 \text{ otherwise.}$$

Step 7: Dual Solution. Compute a dual solution, v_D, as the sum of the costs of the optima in Steps 5 and 6. If $v_D > v_D^*$ save a new dual incumbent $v_D^* \leftarrow v_D$.

Step 8: Primal Solution. Create a feasible solution to (DC) by paying full fixed charges on any arc used in the flow of Step 5. Let v_P be its cost. If $v_P < v_P^*$, save a new primal incumbent as an approximate optimum to (FC). If not, check whether the rate of improvement in the ratio v_P^*/v_D^* is satisfactory. If so, go to Step 10.

Step 10: Dual Update. Modify duals $\sigma_{ij}[k]$, and $\delta_{ij}[\ell]$ by taking a finite step along a subgradient of the Lagrangean dual function at the current dual point and projecting to restore nonnegativity (see for example Bazaraa and Goode (1979) for details on subgradient schemes). Then return to Step 2.

Step 11: Disaggregation. Further subdivide the present artificial commoditization of $S \times \mathcal{D}$. Add any newly created distinct supply subset A_k to $\mathcal{A}$ and pick an appropriate nonnegative starting value for corresponding dual variables $\{\sigma_{ij}[k]:(i,j)\in E\}$. Similarly, add newly created distinct demand subsets B_ℓ to $\mathcal{B}$ and choose nonnegative $\{\delta_{ij}[\ell]:(i,j)\in E\}$. Then, return to Step 2.

4. Artificial Commodity Structures

One important set of issues surrounding the implementation of the above algorithm concerns the family of artificial commodity structures employed. The algorithm is impacted by commodity structure in several ways.

- Relaxation Tightness. One aspect is the degree to which the linear programming relaxation of problem (9) - (16), (19), (20) tightly

approximates the underlying integer problem. Commodities impact relaxation tightness through the fact that there is one set of constraints (19) for each distinct supply set (i.e. each $A_k \varepsilon A$) and one set of constraints (20) for each distinct demand set (each $B_\ell \varepsilon B$). Relaxations associated with different commodity structures differ only in the limitations imposed by these constraints.

- Dual Variables. The number of dual variable sets $\{\sigma_{ij}[k]:(i,j)\varepsilon E\}$ and $\{\delta_{ij}[\ell]:(i,j)\varepsilon E\}$ which must be stored and searched over in any commoditization is also controlled by the dimension of the distinct supply and demand subset sets A and B. For each $A_k \varepsilon A$ and each $B_\ell \varepsilon B$ there is a set of constraints (19) or (20) and an associated set of dual variables.

- Shortest Path Problems. Step 3 of the algorithm calls for finding shortest paths between all supply-demand pairs. Arc lengths $\tilde{v}_{ij}[\alpha,\beta]$ for shortest path problems are as in (30). Assume, as is usually the case, that there are many fewer supply nodes than demand nodes (Symmetric arguements could be given for the opposite case). Then, noting all $\tilde{v}_{ij}[\alpha,\beta]$ are maintained nonnegative throughout processing, a version of the efficient Dijkstra (1959) algorithm should be employed to compute shortest paths. But the Dijstra algorithm can compute simultaneously the shortest path from one node to all other nodes. Thus, if $\tilde{v}_{ij}[\alpha,\beta]$ is independent of β, the Dijstra procedure needs to be invoked only once per $\alpha \varepsilon S$. However, if the $\sum(\delta_{ij}[\ell]/d[\ell])$ term of (30) creates different $\tilde{v}_{ij}[\alpha,\beta]$, the procedure must be applied once per $\alpha \varepsilon S$ and per demand subset with distinct $\tilde{v}_{ij}[\alpha,\beta]$. In total

$$|S| \cdot \begin{pmatrix} \text{number of combinations of} \\ B_\ell \varepsilon B \text{ to which any } \beta \\ \text{simultaneously belongs} \end{pmatrix} \tag{33}$$

shortest path will be required per execution of Step 3.

From the above it is clear that all impacts of artificial commodity structure are controlled by the supply subset list $A \triangleq \{A_k\}$ with each $A_k \subset S$ and the demand subset list $B \triangleq \{B_\ell\}$ with each $B_\ell \subset \mathcal{D}$. To compare possibilites, define a structure $[\tilde{A},\tilde{B}]$ to be tighter than another $[A,B]$

$$\left\{\begin{array}{l} y_{ij} \text{ satisfying (19) for } \tilde{A}_k \varepsilon \tilde{A} \\ \text{and (20) for } \tilde{B}_\ell \varepsilon \tilde{B} \end{array}\right\} \subset \left\{\begin{array}{l} y_{ij} \text{ satisfying (19) for } A_k \varepsilon A \\ \text{and (20) for } B_\ell \varepsilon B \end{array}\right\}$$

That is, $[\tilde{A},\tilde{B}]$ is tighter than $[A,B]$ if it provides at least as tight a linear programming relaxation. We can then obtain some simple dominance results.

Lemma 2: Dominance of Covering Subsets. Let $[A,B]$ be a commodity structure for flows in $S \times \mathcal{D}$, i.e. A a list of distinct nonempty subsets of S and B a similar list of subsets of $\mathcal{D}$. Also, define $\bar{A} \subset S - \underset{A}{\cup} A_k$ and $\bar{B} \subset \mathcal{D} - \underset{B}{\cup} B_k$. Then both $[A \cup \{\bar{A}\}, B]$ and $[A, B \cup \{\bar{B}\}]$ are tighter than $[A,B]$. Also, $[A \cup \{\bar{A}\}, B \cup \{\bar{B}\}]$ is tighter than either $[A \cup \{\bar{A}\}, B]$ or $[A, B \cup \{\bar{B}\}]$. That is, extending the parts of S and $\mathcal{D}$ covered by A and B tightens the formulation.

Proof: Immediate from the fact that new constraints (19) for $\bar{A}$ and/or (20) for $\bar{B}$ are added, without deleting any others.

■

Lemma 3: Dominance of Partitioning Subsets. As above let $[A,B]$ be a commodity structure for flow in $S \times \mathcal{D}$, and pick any $A_k \varepsilon A$ such that $\sum_{\alpha \varepsilon A_k} s_\alpha \leq \sum_{\beta \varepsilon \mathcal{D}} d_\beta$ and any $B_\ell \varepsilon B$ with $\sum_{\beta \varepsilon \mathcal{D}} d_\beta \leq \sum_{\alpha \varepsilon S} s_\alpha$. Then both $[\hat{A},B]$, and $[A,\hat{B}]$ are tighter than $[A,B]$ and $[\hat{A},\hat{B}]$ is tighter than $[\hat{A},B]$ or $[A,\hat{B}]$ where

$$\hat{A} = A - \{A_k\} \cup \{A_k^i: \text{ all } A_k^i \subset A_k,\ A_k^i \cap A_k^j = \phi \text{ for } i \neq j, \cup_i A_k^i = A_k\} \tag{34}$$

$$\hat{B} = B - \{B_\ell\} \cup \{B_\ell^i: \text{ all } B_\ell^i \subset B_\ell,\ B_\ell^i \cap B_\ell^j = \phi \text{ for } i \neq j, \cup_i B_\ell^i = B_\ell\} \tag{35}$$

That is, replacing such A_k and B_ℓ by a partition of them yields a tighter relaxation.

Proof: We shall show only the case of $[\hat{A},B]$ tighter than $[A,B]$ where $\hat{A} = A - \{A_k\} \cup \{A_k^1\ A_k^2\}$ with $A_k^1 \subset A_k$, $A_k^2 \subset A_k$, $A_k^1 \cap A_k^2 = \phi$, and $A_k^1 \cup A_k^2 = A_k$. All other cases follow by analogous arguement for $\hat{B}$ and straightforward induction on the number of $\{A_k^i\}$ or $\{B_\ell^i\}$ respectively.

For our case the only difference in formulations $[A,B]$ and $[\hat{A},B]$ is the former contain

$$\frac{\sum_{\alpha \in A_k} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k} s_\alpha, \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{36}$$

versus the latter's

$$\frac{\sum_{\alpha \in A_k^1} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k^1} s_\alpha, \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{37}$$

and

$$\frac{\sum_{\alpha \in A_k^2} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k^2} s_\alpha, \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{38}$$

By the hypothesis that $\sum_{\alpha \in A_k} s_\alpha \leq \sum_{\beta \in \mathcal{D}} d_\beta$, the supply sum provides the minimum, in denominations of (36) - (38). Thus, noting A_k^1 and A_k^2 partition A_k, the proof reduces to showing

$$\max\left\{\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha},\ \frac{\sum_{\alpha\in A_k^2}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^2} s_\alpha}\right\} \geq$$

$$\left\{\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta] + \sum_{\alpha\in A_k^2}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha + \sum_{\alpha\in A_k^2} s_\alpha}\right\} \tag{39}$$

Assume the A_k^1 term provides the max on the left in (39). Then if (39) fails

$$\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha} < \frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta] + \sum_{\alpha\in A_k^2}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha + \sum_{\alpha\in A_k^2} s_\alpha}$$

Cross multiplying and simplifying leads to

$$\left[\sum_{\alpha\in A_k^2} s_\alpha\right]\left[\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]\right] < \left[\sum_{\alpha\in A_k^1} s_\alpha\right]\left[\sum_{\alpha\in A_k^2}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]\right]$$

or

$$\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha} < \frac{\sum_{\alpha\in A_k^2}\sum_{\beta\in\mathcal{D}} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^2} s_\alpha}$$

Since this contradicts the assumption that A_k^1 provides the maximum in (39), we can conclude (39) holds, and the Lemma follows.

■

Lemma 2 makes it clear that tighter relaxations will result if we consider only commodity structures with $[A,B]$ covering $[S,\mathcal{D}]$, i.e.

$$\bigcup_{A} A_k = S \tag{40}$$

$$\bigcup_{B} B_\ell = \mathcal{D} \tag{41}$$

There could, of course, be a price in terms of dual variables and shortest path problems for demanding a cover. However, at most one new supply group $S - \bigcup_{A} A_k$, and one new demand group $\mathcal{D} - \bigcup_{B} B_\ell$, would have to be added to a non-covering $[A,B]$. Thus, only two new sets of dual variables and perhaps no new shortest path calculations are implied. For these reasons we enforce (40) and (41) in all further discussion.

We shall also demand commodity structures be nonoverlapping i.e.

$$A_k \cap A_i = \emptyset \qquad \text{for all } A_k, A_i \in A,\ i \neq k \tag{42}$$

$$B_\ell \cap B_j = \emptyset \qquad \text{for all } B_\ell, B_j \in B,\ j \neq \ell \tag{43}$$

Lemma 3 provides part of the arguement for the latter restrictions. That lemma shows relaxations are usually tightened when a supply set A_k (or a demand set B_ℓ) is partitioned. It also follows, for example, that when $A_1 \subset A_2$, it is preferable to include sets A_1 and $A_2 - A_1$ in the commodity structure instead of A_1 and A_2. We see that there is usually a gain in relaxation tightness when supply or demand sets do not overlap. In the $A_1 \subset A_2$ example there was not even an increase in dual variables. However, replacing an arbitrary A_1 and A_2 by $(A_1 - A_2)$, $(A_1 \cap A_2)$, and $(A_2 - A_1)$ would tighten the relaxation only by a net increase of one system of dual variables.

The other arguement for nonoverlapping sets as in (42) and (43) relates to the number of shortest path problems (33). Since subsets in any list $\mathcal{B}$ are distinct, (33) cannot be less than $|S| \cdot |\mathcal{B}|$. Any $\mathcal{B}$ satisfying (43) achieves that lower limit.

5. Implementation Issues

Based on the analysis of the previous section, we propose to implement the Lagrangean relaxation algorithms of Section 3 via supply group and demand group lists $\mathcal{A}$ and $\mathcal{B}$ that always partition S and $\mathcal{D}$ as in (40) - (43). Step 1 will create initial partitions, and each time disaggregation Step 11 is executed either some $A_k \in \mathcal{A}$ will be replaced by two nonoverlapping sets A_k^1 and A_k^2, or some $B_\ell \in \mathcal{B}$ will be replaced by a similarly partitioning pair B_ℓ^1, B_ℓ^2.

Even within this approach to disaggregation, there remain many issues regarding implementation of the algorithm of Section 3. When the algorithm starts, a decision must be made with regard to the initial number of subsets in $\mathcal{A}$ and $\mathcal{B}$ and the elements of each of these subsets. Then, at every iteration it must be decided whether to further the disaggregation by partitioning an $A_k \in \mathcal{A}$ or $B_\ell \in \mathcal{B}$. When the decision to proceed with the disaggregation is made, a series of additional decisions are confronted, including selection of the subset to be partitioned, the assignment of its elements to the new subsets, and the initialization of the dual variables corresponding to the new subsets.

5.1 Initial Generation of Artificial Commodities

At the start of the procedure it could be decided to have one or more elements in partitioning lists $\mathcal{A}$ and $\mathcal{B}$. If the decision is to start with singletons $\mathcal{A} = \{S\}$, $\mathcal{B} = \{D\}$, all further partitioning of the original source node set and the original destination node set will be performed in the disaggregation Step 11.

An alternative is to partially partition S and D from the beginning. In general more dual variables and more shortest path problems will result in early iteration. However, if the source nodes and the destination nodes are initially grouped based on a careful analysis of the problem to be solved, the relaxation may be much tighter so that progress on the dual bound in the initial stages of the procedure is faster, favorably compensating the additional computational burden brought on by handling more artificial commodities from the beginning. It is also possible that by starting from an "intelligent" list of supply and demand subsets, further disaggregation of these initial subsets would be more beneficial because the initial grouping has already considered concerns too bulky to include each disaggregation step. Finally, an initial subdivision of S and D obviously implies the number of times the disaggregation step will later be invoked by the algorithm is significantly reduced. Thus results may be less sensitive to the effectiveness and efficiency of Step 11 calculations.

In light of these potential advantages non-singleton initial disaggregations are being tried in computational testing presently underway. In picking initial groups the goal is to quickly reach a tight relaxation without producing too many elements of the initial A and B lists. Noting the form of constraints (19) and (20) it appears we would like to segregate supply and demand points with large s_α and d_β respectively. Otherwise, their presence in the denominator of (19) or (20) dilutes the impact of other flows on y_{ij}. Similarly, if a node is isolated, and thus particularly expensive to service, it seems reasonable to employ a strong relaxation in regard to it, i.e. isolate it in a separate supply or demand set.

For these reasons the initial disaggregation Step 1 being tested automatically segregates in one-point sets any supplies and demands with unusually high servicing cost or supply/demand. For remaining supply and demand points,

constraints (19) and (20) will be strongest if flows tending to have a common shortest supply-demand path are grouped. In the algorithms initial groups are formed so that ones with the most common path elements are together.

Figure 2 shows a single-supply example of these initial disaggregation notions. Since there is only one supply, $A = S = \{1\}$. The initialization rules we have outlined would create a starting partition of $\mathcal{D} = \{2,4,5,6,7,8\}$ as $\mathcal{B} = \{\{5\}, \{6\}, \{2,4\}, \{7,8\}\}$. Node 5 is isolated because of its high demand, node 6 because arcs entering it are particularly costly. Among the remaining nodes, 2 is placed with 4 because all paths to 4 pass through 2, and 7 with 8 because many paths to 8 transit 7.

5.2 Selections of the New Partition

In the dual ascent procedure, used in conjunction with the progressive disaggregation procedure described herein, whenever the rate of improvement on the bound of the optimal solution to $(P_{\sigma\delta}[A,\mathcal{B}])$ does not meet the minimum standards set beforehand, it signals the need to further disaggregate some of the current artificial commodities. This is carried out by partitioning one or more supply and/or destination node subsets. As noted above we have chosen to partition only one subset at any one time. The main reason for such choice is to keep the procedure simple while still achieving the goals of the disaggregation.

The selection of the subset to be partitioned involves ranking the current subsets according to some criterion that matches our strategic objective -- significiant improvement of the dual bound. As we have explained earlier, the disaggregation pattern affects the dual bound only through constraints (19) and (20). In the algorithm of Section 3, those constraints are included in the $(P_{\sigma\delta}[A,\mathcal{B}])$ objective function as terms

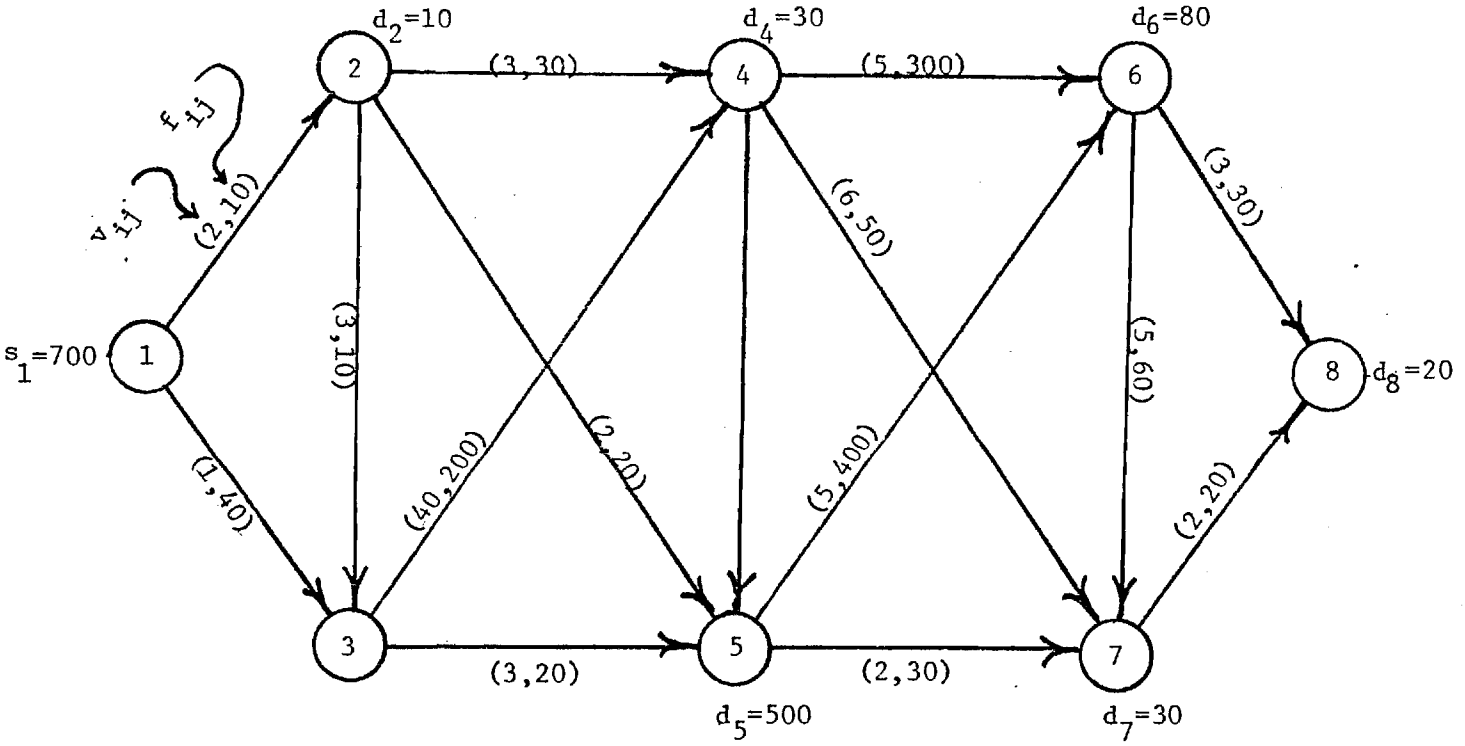


Figure 2. Initial Disaggregation Example

$$\sum_{(i,j)\in E} \sigma_{ij}[k]\left[\left(\frac{f_{ij}}{m[k]}\right)\left(\frac{\sum_{\alpha\in A_k}\ \sum_{\beta\in D} x_{ij}[\alpha,\beta]}{s[k]}\right) - y_{ij}\right] \tag{44}$$

and

$$\sum_{(i,j)\in E} \delta_{ij}[\ell]\left[\left(\frac{f_{ij}}{n[\ell]}\right)\left(\frac{\sum_{\beta\in B_\ell}\ \sum_{\alpha\in S} x_{ij}[\alpha,\beta]}{d[\ell]}\right) - y_{ij}\right] \tag{45}$$

where $s[k]$ and $d[\ell]$ are as in (31) and (32). One new element is nonnegative weights $(f_{ij}/m[k])$ and $(f_{ij}/n[\ell])$ used to scale constraints (19) and (20) for greater subgradient search efficiency. Generally, $m[k]$ is similar in magnitude to $s[k]$, and $n[\ell]$ to $d[\ell]$.

Since the expressions in (44) and (45) are less than or equal to zero in feasible solutions, minimizing their absolute value will tend to improve the dual bound quality. Consequently, we select for partition the subset for which the corresponding expression (44) or (45) is the most negative. The implementation of this selection rule is very simple and it does not involve any additional calculations, since the values of expressions (44) and (45) are always readily available in the dual ascent procedure where they are used in evaluating the objective function.

Once the subset to be partitioned is identified, it is necessary to determine how to partition it. This includes deciding how many new subsets to create and which elements of the subset being partitioned to assign to the new subsets.

With regard to the composition of the two new subsets, a criterion similar to the one used in selecting the subset to be partitioned is applied. For each element of the selected subset, its contribution to the expression in (44) for a source node subset, or to (45) for a destination node subset, is evaluated. Based on these contributions, the elements with the highest contributions will be assigned to one of the subsets, and the rest of the elements will be assigned to the other. Each of the new subsets is required to have the same number of elements, so that all singletons will be reached in the minimum number of partitions. Again, these decision rules are quite simple to implement because (44) and (45) are readily available.

5.3 Initializing Dual Variables

Once it has been decided to partition a supply group A_k or demand group B_ℓ, initial values must be chosen for dual varialbes $\sigma_{ij}[k]$ or $\delta_{ij}[\ell]$ and for scaling coefficients $m[k]$ or $n[\ell]$. We shall discuss the case of partitioning a demand set B_ℓ into two new sets B_p and B_q for which we seek new duals $\{\delta_{ij}[p]$ and $\delta_{ij}[q]: (i,j)\varepsilon E\}$ and scaling weights $n[p]$ and $n[q]$. The case of partitioning a supply subset A_k is completely analogous.

In the previous section we showed how the goal in selecting B_p and B_q was one of maximizing the short term improvement in dual bound. We would, of course, like initial dual variables to also advance the dual solution. But there is another important issue: we desire stability in the dual search so that any poorly chosen duals will quickly be corrected by Step 10 of the algorithm.

To obtain stability, we seek to assure that the x and y primal solutions of Steps 5 and 6 of the algorithm (Section 3) will not decrease violently in the first iteration after disaggregation. (If group selection was sound the dual value should improve).

At Step 6, $y_{ij} \leftarrow 1$ if $\hat{f}_{ij} \leq 0$ and 0 otherwise, where (including the scaling factor $f_{ij}/n[\ell]$)

$$\tilde{f}_{ij} = f_{ij}\left[1 - \sum_{A_k \varepsilon A} \sigma_{ij}[k]/m[k] - \sum_{B_\ell \varepsilon B} \delta_{ij}[\ell]/n[\ell]\right] \tag{46}$$

Dividing B_ℓ into B_p and B_q in the B list will merely replace

$$\frac{\delta_{ij}[\ell]}{n[\ell]} \text{ with } \frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]}$$

Thus, the y_{ij} solution will be unchanged if

$$\frac{\delta_{ij}[\ell]}{n[\ell]} = \frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]} \tag{47}$$

To similarly preserve the $x_{ij}[\alpha,\beta]$ solution of algorithm Steps 3-5, we desire to leave unchanged shortest path arc lengths

$$\tilde{v}_{ij}[\alpha,\beta] = v_{ij} + f_{ij}\left[\sum_{\{A_k \varepsilon A:\alpha \varepsilon A_k\}} \frac{\sigma_{ij}[k]}{m[k]s[k]} + \sum_{\{B_\ell \varepsilon B:\beta \varepsilon B_\ell\}} \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}\right] \tag{48}$$

After partition each β will belong to $\mathcal{B}_p$ or $\mathcal{B}_q$, but not both. Thus, either

$\frac{\delta_{ij}[p]}{n[p]d[p]}$ or $\frac{\delta_{ij}[q]}{n[q]d[q]}$ will replace $\frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$ in (48),

The dual selection we propose is fixing

$$n[p] \leftarrow n[\ell] \tag{49}$$

$$n[q] \leftarrow n[\ell] \tag{50}$$

$$\delta_{ij}[p] \leftarrow \delta_{ij}[\ell] \frac{d[p]}{d[\ell]} \tag{51}$$

$$\delta_{ij}[q] \leftarrow \delta_{ij}[\ell] \frac{d[q]}{d[\ell]} \tag{52}$$

Substitution in (47) gives

$$\frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]} = \frac{\delta_{ij}[\ell]}{n[\ell]} \left[\frac{d[p]+d[q]}{d[\ell]}\right] = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

the last because $\mathcal{B}_p$ and $\mathcal{B}_q$ partition demands in $\mathcal{B}_\ell$. Also, (49)-(52) yield

$$\frac{\delta_{ij}[p]}{n[p]d[p]} = \frac{\delta_{ij}[\ell]}{n[\ell]d[p]} \left(\frac{d[p]}{d[\ell]}\right) = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

and

$$\frac{\delta_{ij}[q]}{n[q]d[q]} = \frac{\delta_{ij}[\ell]}{n[\ell]d[q]} \left(\frac{d[q]}{d[\ell]}\right) = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

as required to preserve the $\tilde{v}_{ij}[\alpha,\beta]$ of (48).

6. Experimentation

Previous sections outlined the development of a strategy for implementing progressive disaggregation in the context of a Lagrangean relaxation algorithm for tight formulations of fixed charge network flow problems. Justifications provided for details of the algorithm do consider problem properties, but their true effectiveness can only be measured empirically. Thus, a series of experiments involving variants of these strategic decisions is presently underway.

REFERENCES

Bazaraa, M.S. and J.J. Goode (1979) "A Survey of Various Tactics for Generating Lagrangian Multipliers in the Context of Lagrangian Duality," European Journal of Operations Research, 3, 322-338.

Dijkstra, E.W. (1959), "A Note on Two Problems in Connexion with Graphs," Numer Math, 1, 269-271.

Rardin, R.L. (1982), "Tight Relaxations of Fixed Charge Network Flow Problems," Industrial and Systems Engineering Report Series No. J-82-3, January.

Rardin, R.L. and Ui Choe (1979), "Tighter Relaxations of Fixed Charge Network Flow Problems," Industrial and Systems Engineering Report Series J-79-18, May.

ATTACHMENT 3

Industrial and Systems Engineering
Report Series J-82-2
March, 1982

SOME POLYNOMIALLY SOLVABLE MULTI-COMMODITY FIXED CHARGE NETWORK FLOW PROBLEMS

by

Ronald L. Rardin
R. Gary Parker

and

Wang Kyu Lim

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS - 8018954

ABSTRACT

Recent developments have shown that many uncapacitated multi-commodity fixed charge network flow problems admit very tight linear programming relaxations in the sense that continuous solutions closely approximate discrete ones. In this paper we show that on series-parallel graphs those linear programming relaxations are perfect, i.e. they yield discrete optima. We also illustrate how a number of known combinatorial optimization problems can be formulated in this fixed charge format. The implication is that all the indicated fixed charge problems, including the specified combinatorial cases, are solvable in polynomial time via linear programming.

1. Introduction

The Uncapacitated p-Commodity Fixed Charge Network Flow Problem can be stated

$$\min \quad \sum_{k=1}^{p} \hat{v}^k \hat{x}^k + f x^0 \tag{1-1}$$

$$\text{s.t.} \quad E\,\hat{x}^k = \hat{r}^k \qquad \text{for } k=1,2,\ldots,p \tag{1-2}$$

$$x^k \geq 0 \qquad \text{for } k=1,2,\ldots,p \tag{1-3}$$

$$\rho\, x_j^0 \geq \sum_{k=1}^{p} x_j^k \qquad \text{for } j=1,2,\ldots,n \tag{1-4}$$

$$x_j^0 = 0 \text{ or } 1 \qquad \text{for } j=1,2,\ldots,n \tag{1-5}$$

where E is the vertex-arc incidence matrix of an n-arc directed network, $\hat{x}^k$ is the flow vector of commodity k in the network, $\hat{v}^k$ is the vector of variable costs per unit for flow $\hat{x}^k$, $\hat{r}^k$ is a requirements vector on commodity k, f_j is a fixed cost incurred when at least one commodity uses arc j, x^0 is a vector of 0-1 variables switching "on" and "off" the fixed charges, ρ is a large constant, and 0 and 1 represent appropriate vectors or matrices of 0's and 1's respectively. We assume throughout that $f \geq 0$ and that the sum of $\hat{v}_j^k$ around any directed cycle in the commodity k network is nonnegative. Under these assumptions, $x_j = 1$, and f_j is paid, exactly when constraint (1-4) allows flow.

Unlike the usual multi-commodity flow problem (see for example Kennington and Helgason (1980)) commodities do <u>not</u> compete for an arc capacity. Instead, they interact through the shared fixed installation/construction/setup cost, f_j.

We term the problem "single-source" because we shall assume that each commodity is supplied at a single vertex, although this vertex may vary with commodity. This assumption implies that each $\hat{r}^k$ is a vector with components summing to zero which has a unique positive component at the supply point, negative entries at various demand points, and zero entries where the commodity is only transhipped. Of course, any uncapacitated commodity flow problem can be placed in single source form by adding an "super source" if required. However, this transformation would change the form of the graph--the issue to which we will shortly direct our interest.

Although formulation (1-1) - (1-5) is a correct mixed-integer statement of the problem, previous work (see Rardin and Chou (1979)) has shown that its linear programming relaxation (replacing (1-5) by $0 \leq x^0 \leq 1$) often gives a very poor approximation to an optimal solution. Much better results are obtained if each commodity is subdivided into separate commodities for each demand point. One can then rescale flows, requirements, and variable costs so that all demands are unity. The result has the form

$$\min \quad \sum_{k=1}^{q} v^k x^k + fx^0 \tag{1-6}$$

(MCFC) s.t. $\quad Ex^k = r^k \qquad$ for $k=1,2,\ldots,q$ (1-7)

$$x^0 \geq x^k \geq 0 \qquad \text{for } k=1,2,\ldots,q \tag{1-8}$$

$$x^0 \text{ integer} \tag{1-9}$$

with each r^k having exactly one +1 and one -1, and q the revised number of commodites.

In this paper we investigate cases for which the linear programming relaxation of (MCFC) is "perfect", i.e. cases for which the integrality requirement (1-9) is redundant. In Section 2 we use results in Truemper (1977) and Truemper and Soun (1979) to show that if the graph associated

with matrix E has a certain series-parallel property, (1-6) through (1-8) will always solve in integers. Thus, since linear programming problems can be solved in polynomially-bounded time (Khachian (1979)), it follows that all such (MCFC) are polynomially solvable. Section 3 shows how several familiar combinatorial optimization problems can be formulated as (MCFC), in which case, they too are polynomially solvable on the indicated graphs.

2. Unimodular Cases

A matrix M can be termed unimodular if the determinant of every maximal nonsingular square submatrix $\bar{M}$ (denoted $det\ (\bar{M})$) is ± 1. Unimodularity is a weaker property than total unimodularity for which all square submatrices $\bar{M}$ have $det\ (\bar{M}) = 0, \pm 1$. Still, it is well known (see Vienott and Dantzig (1968)) that a linear constraint system

$$Ax = b$$

$$x \geq 0$$

has integer extreme points if A is unimodular, and A and b are integer; basic solutions computed by Cramer's rule will have unit denominators.

Suppose E is the vertex-arc incidence matrix of a directed graph, T. Thus, if arc $a_k = (i,j)$ belongs to T, column k of E has a +1 in row i and a -1 in row j. Such E are well known to be totally unimodular. Truemper (1977) investigated conditions under which constraints $Ex = b$ may be supplimented with additional linear constraints $Dx + s = d$ while keeping the corresponding constraint matrix at least unimodular.

A subgraph of a graph, say $\bar{T}$, is said to be Euler if every vertex of $\bar{T}$ is incident to an even number of arcs of $\bar{T}$. A vector $\bar{x}$ is a circulation vector on a subgraph $\bar{T}$ of T if $E\bar{x} = 0$ and $\bar{x}_j = 0$ whenever the corresponding edge j of T does not belong to $\bar{T}$. In terms of these definitions, Truemper's main result can be stated as follows:

Lemma 1. Unimodularity of Networks with Side Constraints (Truemper (1977)). Let E be the vertex-arc incidence matrix of a directed graph T and D a matrix of integers. Then the matrix

$$\begin{bmatrix} E & 0 \\ D & I \end{bmatrix}$$

is unimodular if and only if

(i) There exists a spanning forest of T such that for every cycle C^k formed by adding arc k to the forest there is a nonzero circulation vector x^k with 0, ± 1 components such that Dx^k is a vector with 0, ± 1 components; and

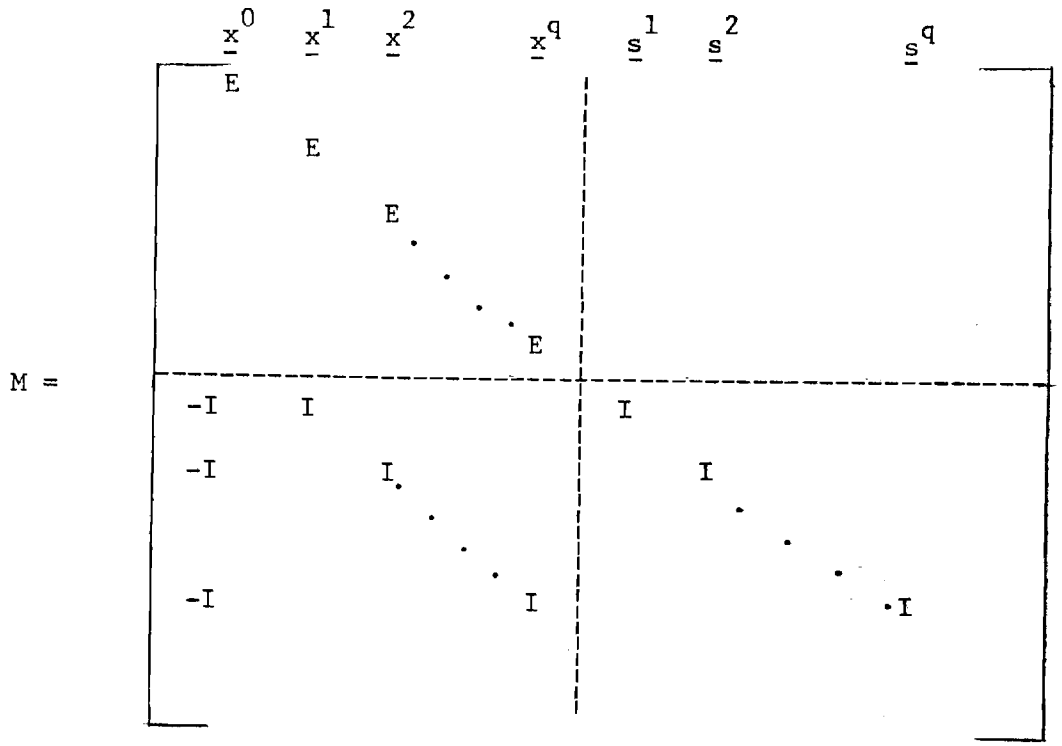
(ii) For every Euler subgraph $\bar{T}$ of T with nonempty arc set, there is a nonzero circulation $\bar{x}$ on $\bar{T}$ such that components $(D\bar{x})_i = 0$ whenever row sum $(Dl)_i$ is even.

■

Our interest here is in the unimodularity of the constraint system for problem (MCFC). As stated in (1-7) and (1-8), the vector x^0 is not subject to balance of flow restrictions,

$$Ex^0 = r^0 \qquad (2\text{-}1)$$

However, it will be convenient to add such constraints for the moment. Now when slacks, s^k, are also added in the constraints (1-8), the matrix of interest becomes



The graph, T, of Lemma 1 is simply q+1 copies of the directed graph G over which (MCFC) is defined.

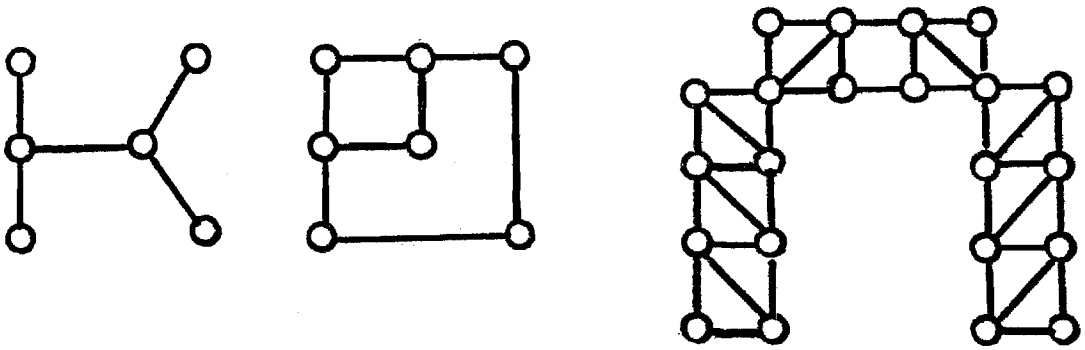
A connected, undirected graph U is said to be series-parallel if it can be reduced to a tree by sequential application of the following operations:

series reduction: Replace a degree-2 vertex j and incident edges (i,j) and (j,k) by a single edge (i,k).

parallel reduction: Eliminate one of any two parallel edges connecting the same pair of vertices.

Observe that the operation of these equations is well-defined (i.e. unambiguous). An equivalent definition (see Duffin (1965)) is that U is series-parallel if and only if it possesses no subgraph that can be reduced to K_4 (the complete graph on 4 vertices) by using only series reduction. Clearly, the latter assures all series-parallel graphs are planar, but planar graphs need not be series parallel. Some useful examples are given in Figure 1.

(a) Some Series-Parallel Graphs



(b) Some Planar, but Non-Series-Parallel Graphs

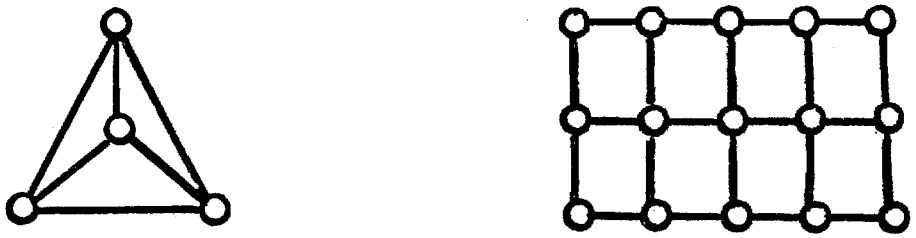


Figure 1: Examples of Series-Parallel and Non-Series-Parallel Graph

With these definitions we can now state our main result.

Theorem 2: Unimodularity of Series Parallel (MCFC) with x^0 Flow Balance:
Let E be the vertex-arc incidence matrix of a directed graph G. Then the undirected graph obtained by ignoring direction in G is series-parallel if and only if every matrix of the above form M is unimodular.

Proof: We shall apply Lemma 1. The graph T of that lemma is the union of (q+1) disjoint copies of the G for this theorem. Thus, any spanning forest will include a spanning forest from each of the (q+1) components. Also, cycles of Lemma 1 part (i) will belong entirely to one component. Since the lower, D-section of matrix M consists entirely of zero and identity sub-matrices corresponding to the components,it follows that any 0, +1, -1 circulation on a cycle will yield a 0, +1, -1 total below. This establishes condition (i) of Lemma 1.

To show the series-parallel property is necessary to condition (ii) of the lemma, consider the K_4 example of Figure 2, and pick the 4-edge cycles 4-1-3-2-4 and 4-1-2-3-4 as Euler Subgraphs in the x^0 and x^1 commodities respectively. The two cycles share arcs (4,1) and (2,3), and condition (ii) will be satisfied only if we can find a nonzero circulation in the two commodities that agrees on the two arcs. However, it is easy to check that any circulation (i.e. weighting which sums to zero at each vertex) must have x^0 weights on (4,1) and (2,3) with opposite signs, while such a circulation for the x^1 subgraph must have matching weights on the two arcs.

To prove the series-parallel property is sufficient for property (ii) of Lemma 1, we proceed inductively by reversing the defining reductions of series-parallel graphs. Property (ii) holds trivially for trees since trees have no Euler subgraphs with nonempty edge sets. By definition of series-parallel graphs, more complex cases can be reduced to trees by a sequence of series and parallel reductions. Thus, by reversing the

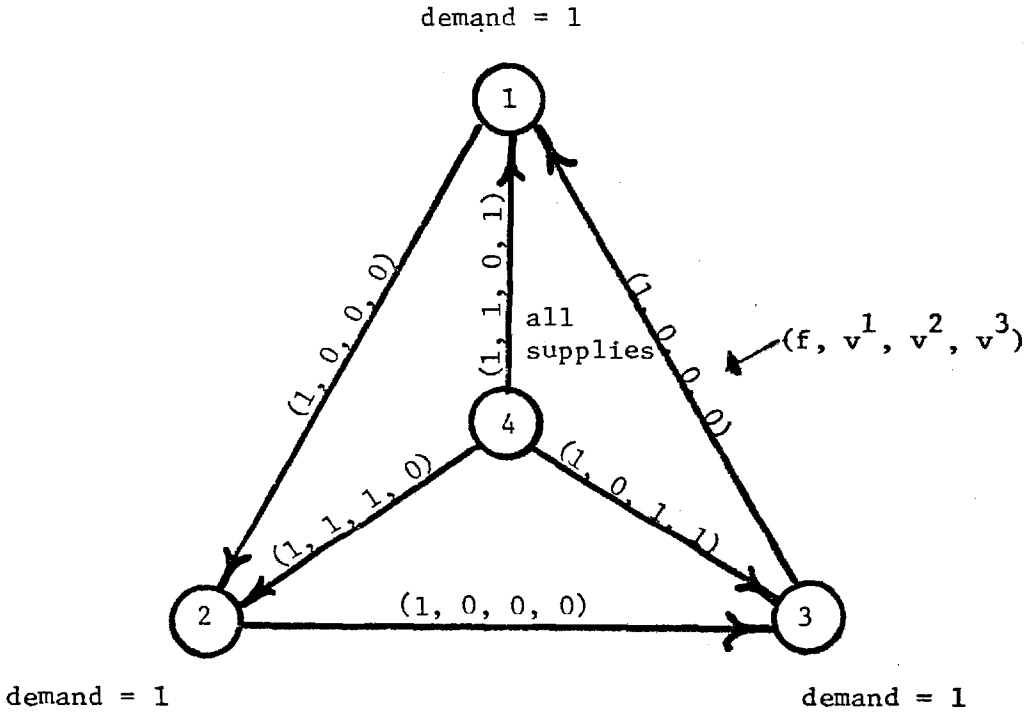


Figure 2: Example Failing Unimodularity When Not Series-Parallel

sequence of such reductions, any series-parallel graph can be reconstructed from a tree. We shall show that if a given graph $\bar{G}$ satisfies property (ii), any graph, G, obtained from $\bar{G}$ by reversing either a series or a parallel reduction also satisfies the property. Thus, inductively, the property will be seen to hold for all series-parallel graphs.

Assume that property (ii) holds for a present graph $\bar{G}$ and let G be the next graph in this reverse reduction order. Since the graph, T, of Lemma 1, property (ii) corresponding to G is the union of (q+1) identical copies of G, any Euler subgraph, H, of T will decompose into Euler subgraphs H^0, $H^1,\ldots,H^q$ of the components. The lower D-section of matrix, M, can have even row total $(D1)_i$ only if some arc, e, belongs to both H^0 and at least one of the $\{H^i: i=1,2,\ldots,q\}$.

The case of reversing a series reduction to move from $\bar{G}$ to G is straightforward. Suppose arc $\bar{e} = (i,k)$ of $\bar{G}$ is to be restored to the arc sequence $\{e,f\}$ of G joined at new degree-2 vertex j. For $i=0,1,\ldots,q$ define $\bar{H}^i$ as the graph obtained from H^i by substituting $\bar{e}$ for $\{e,f\}$. Clearly each such $\bar{H}^i$ is an Euler subgraph of $\bar{G}$. Thus, by induction there is a nonzero circulation $\{\bar{x}^i: i=0,1,\ldots,q\}$ on the $\bar{H}^i$ that cancels as required in property (ii). Let $\bar{x}^i_{\bar{e}}$ be the weight in that circulation for edge $\bar{e}$. We need only duplicate it on both of e and f to have a circulation that cancels for the $\{H^i\}$ in G. Specifically, if both e and f have the same direction as $\bar{e}$ we choose $x^i_e \leftarrow x^i_f \leftarrow \bar{x}^i_{\bar{e}}$ for all i. If both have the opposite orientation to $\bar{e}$, we make $x^i_e \leftarrow \bar{x}^i_f \leftarrow -\bar{x}^i_{\bar{e}}$. Similarly, if e and f have opposite directions, we select $x^i_e \leftarrow \bar{x}^i_{\bar{e}}$, $x^i_f \leftarrow -x^i_{\bar{e}}$ or $x^i_f \leftarrow \bar{x}^i_{\bar{e}}$, $x^j_e \leftarrow \bar{x}^i_{\bar{e}}$.

Now consider reversing a parallel reduction, i.e. adding an arc, e, in G that parallels another , $\bar{e}$, already in $\bar{G}$. We shall assume e and $\bar{e}$ have the same direction because one needs only to reverse the sign of the circulation value for e in the opposed direction case.

As before let $\{H^i: i=0,1,\ldots,q\}$ be (not all empty) subgraphs of the (q+1) copies of G. If H^0 has empty arc set, there is no cancellation to prove. If H^0 has nonempty arc set, we consider three cases:

Case 1: H^0 includes only $\bar{e}$ and e. Create a circulation in H^0 by $x_e^0 \leftarrow +1$, $x_{\bar{e}}^0 \leftarrow -1$. For any other H^i containing both $\bar{e}$ and e, cancel this H^0 circulation by duplicating it. Any Eulerian H^i that contains $\bar{e}$ or e, but not both, necessarily includes a cycle C^i including e (respectively $\bar{e}$). Choose $x_e^i \leftarrow +1$ (respectively $x_{\bar{e}}^i \leftarrow -1$) and then pick ± 1 orientation for other arcs of C^i to form a circulation. The resulting x^i cancels as required on e (respectively $\bar{e}$).

Case 2: H^0 includes $\bar{e}$ and e and other edges of G. For i=0,1,...q, construct Euler subgraphs of $\bar{G}$ as follows:

$$\bar{H}^i \leftarrow \begin{cases} H^i \text{ less } \bar{e} \text{ and } e & \text{if } H^i \text{ contains both } \bar{e} \text{ and } e \\ H^i \text{ less edges in } C^i & \text{if } H^i \text{ contains } \bar{e} \text{ or } e, \text{ but not both} \\ H^i & \text{otherwise,} \end{cases}$$

Here, as above, C^i is any cycle of H^i containing e (or $\bar{e}$).

The $\bar{H}$ systems is Euler because each construction removed a cycle from an Euler subgraph. Moreover, at least $\bar{H}^0$ is nonempty because H^0 had more than e and $\bar{e}$. Thus, the $\bar{H}^i$ are a system of Euler subgraphs of $\bar{G}$ to which property (ii) inductively applies. Let $\{\bar{x}^i\}$ be the implied nonzero circulation. Choosing $x_t^i \leftarrow \bar{x}_t^i$ for all i and t gives the required cancelling circulation for the H^i of G. Zero x_e^i and $x_{\bar{e}}^i$ have been chosen for all i, and thus, cancellation as in property (ii) is achieved. Still, since the $\bar{x}^i$ are not all zero, neither are the x^i.

Case 3: H^0 includes at most one of e and $\bar{e}$. For i=0,1,...,q construct Euler subgraphs of $\bar{G}$ as follows:

$$\bar{H}^i \leftarrow \begin{cases} H^i \text{ less } \bar{e} \text{ and } e & \text{if } H^i \text{ contains both } \bar{e} \text{ and } e \\ H^i \text{ less } e \text{ plus } \bar{e} & \text{if } H^i \text{ contains, } e \text{ but not } \bar{e} \\ H^i & \text{otherwise.} \end{cases}$$

As above the $\bar{H}^i$ are Euler subgraphs because we have only substituted parallel arcs or deleted cycles. Also, the $\bar{H}$ system is not all empty because the edges of $\bar{H}^0$ are the same as those in H^0. Thus, the $\bar{H}$ system is again one to which property (ii) inductively applies in $\bar{G}$. Let $\{\bar{x}^i\}$ be the implied circulation and pick

$$x_t^i \leftarrow \begin{cases} \bar{x}_e^i & \text{if } t=e \text{ and } H^i \text{ contains e but not } \bar{e} \\ 0 & \text{if } t=\bar{e} \text{ and } H^i \text{ contains e but not } \bar{e} \\ \bar{x}_e^0 & \text{if } H^i \text{ contains both e and } \bar{e}, \ t=\bar{e} \\ & \text{and } H^0 \text{ contains } \bar{e} \text{ or } t=e \text{ and } H^0 \\ & \text{contains e} \\ -\bar{x}_e^0 & \text{if } H^i \text{ contains both e and } \bar{e}, \ t=e \\ & \text{and } H^0 \text{ contains } \bar{e} \text{ or } t=\bar{e} \text{ and } H^0 \\ & \text{contains e} \\ \bar{x}_t^i & \text{otherwise} \end{cases}$$

The effect is to shift $\bar{e}$ circulation to e when $\bar{e}$ replaced e in constructing the $\bar{H}$ system. That revised circulation is nonzero because the $\bar{x}$ one was. Also, the circulation must cancel in G because it cancelled on $\bar{e}$ in $\bar{G}$. If the implied circulation was nonzero on the at most one of e and $\bar{e}$ in H^0, we have also balanced it with a circulation on the $\{e, \bar{e}\}$ cycle in each H^i containing that cycle. We conclude that the new circulation $\{x^i\}$ is the one re-

quired for property (ii).

Since Cases 1-3 are exhaustive, the proof is complete.

■

Theorem 2 shows that constraint matrices, M, for versions of (MCFC) also requiring balance of flow in the x^0 variables are unimodular if the associated graph is series-parallel. To see that the result can be extended to the ordinary (MCFC) formulation (1-6) to (1-9) consider the replacing (2-1) by (2-1)

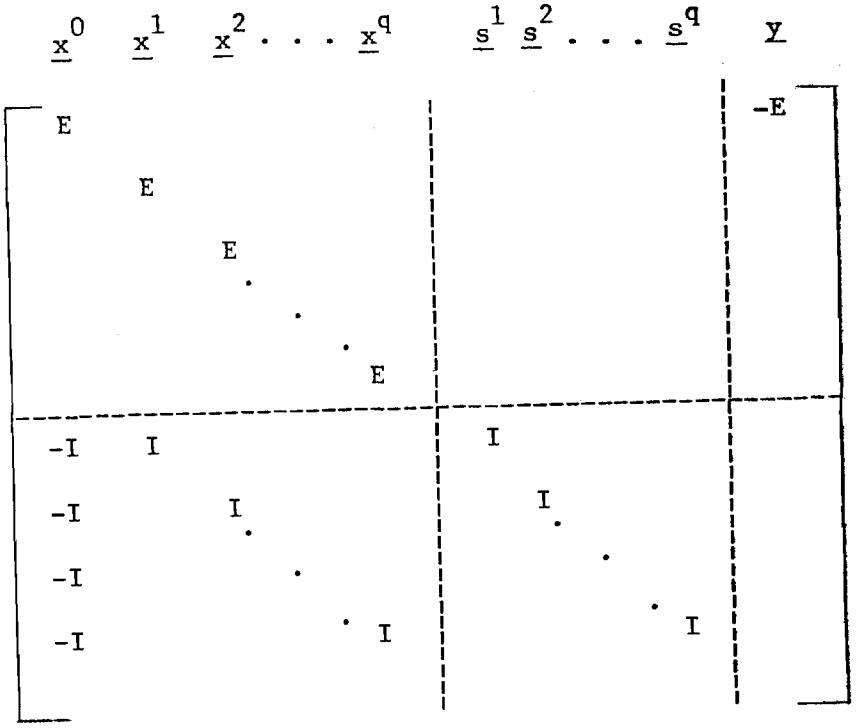
$$Ex^0 - Ey = 0 \tag{2-2}$$

$$y \geq 0 \tag{2-3}$$

Here new zero-cost variables y negate the effect of the x^0 so that a zero balance of flow can be achieved at no cost for any choice of x^0.

The corresponding contraint matrix for this new form is

$N =$



The following corollary shows N is unimodular whenever M is.

Theorem 3: Unimodularity of Series-Parallel (MCFC) Without x^0 Flow Balance. Let E be the vertex-arc incidence matrix of a directed graph G. Then the undirected graph obtained by ignoring direction in G is series parallel if and only if every matrix of the above form N is unimodular.

Proof: The only difference from Theorem 2 is that the x^0/y component of Truemper's graph T no longer matches that of the other x^k. However, it is series-parallel whenever G is because it merely duplicates each arc with one oriented in the opposite direction.

To prove the present theorem we can extend the strategy used for Theorem 2. Necessity is exactly as before. For sufficiency, suppose we rebuild G from a tree in step-by-step order without restoring any y-arcs. All arguements of the proof of Theorem 2 apply. After G is constructed, we add the y-arc system by reversing parallel reductions. Since the new arcs have no coefficients in the lower, D-system of N, no new conflicts of Euler subgraphs will need to be resolved.

■

Theorems 2 and 3 are both necessary and sufficient because the conditions of Lemma 1 are. However, it is conceivable that all non-unimodular bases are dominated or infeasible for (MCFC). The example of Figure 2 shows that if there are at least q=3 commodities, a minimal counter-example is possible. For the indicated cost, the unique optimal solution to the linear programming relaxation of (MCFC) is to make all x^0 variables $\frac{1}{2}$, and to send $\frac{1}{2}$ unit of flow in each demand commodity direct from vertex 4 and the other $\frac{1}{2}$ via the demand point's predecessor in the circuit 1-2-3-1. This solution costs 9/2, while every integer optimum costs 5.

A final note should be added regarding direction. In the proof of our results we have been concerned mainly with the undirected version of our

series-parallel graphs, G. Our only requirement is that whatever orientation is given to an edge, it have the same orientation in all commodities.

It is reasonable to ask whether one could mix orientations with, for example, some commodities having edge ℓ directed (i,j) and others having it directed (j,i). Figure 3 gives a trivial case for which such an extension fails. In order to achieve the cancellation on Euler subgraphs required for Lemma 1 condition (ii), circulations must match on the two k arcs as should those on the two ℓ arcs. Such circulations cannot both sum to zero at the two vertices.

3. Implications for Certain Combinatorial Problems

Results of the previous section show that any (MCFC) on a series-parallel graph can be solved by linear programming. That is, such (MCFC) can be solved in polynomial time. It follows that any problem that can be formulated as (MCFC) (or (MCFC) with flow balance in x^0) is polynomially solvable on series-parallel graphs.

For some combinatorial problems fitting naturally in the fixed charge format the fact that series-parallel cases are polynomially solvable is already known, although not in those general terms. For example, the uncapacitated warehouse location problem (choosing which of several possible "warehouses" to build as sources in a bipartite graph) is unimodular if there are at most 2 sources or 2 sinks (see Cho, Johnson, Padberg and Rao (1981)). Also, many of the one-commodity forms of Erickson (1978) are series-parallel without the "supersink."

However, the opportunity for multi-commodities -- whether natural or artificially induced -- raises many new possibilities. We have outlined below how some typical problems can be placed in (MCFC) form. In each case we believe the result that the problem is polynomially solvable for directed series-parallel graphs is new, although other special cases have been polynomially solved.

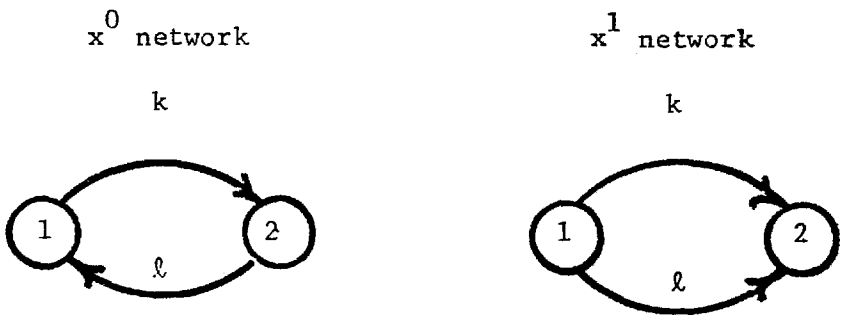


Figure 3. Example Showing Directions Must Match in Commodity Networks

(Readers are referred to Garey and Johnson (1979) for details on particular problem's status). Naturally since all our formulations are weighted, they implicitly include cardinality analogs (i.e. equal weights). Also note that the formulations imply (MCFC) on general graphs is NP-Hard because each problem listed is known to be NP-Hard in the worst case.

In all the definitions to follow we assume G is a directed graph with arc set A and vertex set V. If the given problem is defined on an undirected graph with edge set E, we create a directed case by putting (i,j) and (j,i) in A whenever $(i,j) \in E$. Weights on the two arcs match that on the edge.

Steiner Tree Problem (Nonnegative Edge/Arc Weights).

Problem is to find a minimum total weight (directed or undirected) tree spanning a subset $\bar{V}$ of V. The problem is easily solved when $\bar{V} = V$ (see Lawler (1976)) but generally difficult if $\bar{V} \subseteq V$. To formulate the case where weights are nonnegative as (MCFC), we merely pick some vertex $t \in \bar{V}$ to be the root of the tree (for directed cases all the $|\bar{V}|$ choices of t might have to be tried in turn). All vertices in $\bar{V}-\{t\}$ are then treated as demand points, each with its own commodity. Weights on edges are the fixed costs, f, and all variable costs, v^k, are zero. The optimal Steiner tree will consist of those arcs ℓ for which $x^0_\ell = 1$ in the (MCFC) optimum.

Minimum Equivalent Graph Problem (Nonnegative Edge/Arc Weights) The Minimum Equivalent Graph Problem is to find a minimun total weight system of edges or arcs that includes a directed path between all ordered pairs of vertices $(t,u) \in V \times V$. Clearly, a minimum spanning tree provides an optimum for the undirected case, but the directed case is NP-Hard (on general graphs). To formulate the nonnegative weight directed problem as (MCFC) we need only create one commodity for each pair $(t,u) \in V \times V$. Fixed costs, f, are set to the arc weights and variable costs, v^k, are zero. An optimal solution uses all arcs ℓ for which $x^0_\ell = 1$ in the (MCFC) optimum.

Shortest Total Path Spanning Tree (Nonnegative Edge Weights). The Shortest Total Path Spanning Tree Problem seeks a spanning tree of G for which the sum of the lengths of paths between all paris of vertices is minimal. The problem makes sense only for undirected G. For such G with nonnegative edge weights, the problem can be formed as (MCFC). We adopt virtually the same formulation as the Minimal Equivalent Graph Problem with commodities for each ordered vertex pair. However, this time variable costs, v^k, are set equal to edge weights. Thus, the sum of the variable costs will equal total path length. A spanning tree is the minimal cardinality system connecting all vertices. Thus, we can force the paths to all travel through a tree by making all fixed costs, f_ℓ, equal to a very large constant, ρ. Every tree solution will have $2\ (|V| - 1)$ copies of ρ (both forward and reverse arcs will be needed along the tree). Thus, variable costs will determine optimality. An optimal tree will be formed by choosing edges corresponding to arcs ℓ with $x^0_\ell = 1$ in the (MCFC) optimum.

Minimum Spanning Euler Subgraph (Directed Graphs with Nonnegative Edge Weights). A subgraph $\bar{G}$ of G with vertex and arc sets $\bar{V}$ and $\bar{A}$ is Euler if the number of arcs in $\bar{A}$ directed into each vertex in $\bar{V}$ equals the number directed out. The subgraph is spanning if every vertex of V is joined by some arc of $\bar{A}$. The Minimum Spanning Euler Subgraph Problem seeks the least total weight such subgraph. The nonnegative cost case of this problem is reduced to an (MCFC)-like form in the same general way as the Minimum Equivalent Graph Problem. Commodities are formed for each ordered vertex pair; weights become fixed charges.

The new feature here is that the arcs ℓ with $x^0_\ell = 1$ must form an Euler subgraph. To enforce that requirement we add two new constraint systems:

$$Ex^0 = 0 \tag{3-1}$$

$$x^0 \leq 1 \tag{3-2}$$

System (3-1) assures equal in and out degree, and (3-2) prevents duplicate

use of arcs. Recall that Theorem 2 proved (MCFC) with (3-1) is unimodular. Simple integer upper bounds such as (3-2) cannot change the unimodularity result. The determinant of every basis of the problem including (3-2) is decided by the determinant of an essential basis from matrix M of Theorem 2. (See, for example, Bazaraa and Jarvis (1977) for details).

Travelling Salesman Problem. The famous Travelling Salesman Problem seeks a minimun total length circuit visiting each $t \in V$ exactly once. An optimal solution is a spanning Euler subgraph with one "in" and one "out" arc at each vertex. Thus, it is a minimum spanning Euler subgraph among those of minimum cardinality. To make the above formulation compute a travelling salesman tour, we need only add a large constant, ρ, to each fixed charge. Every feasible tour will incur $|V|$ copies of this constant, so that optimality is determined by total weight. Note that this constant can also assure all costs are nonnegative.

REFERENCES

1. Bazaraa, M.S. and J.J. Jarvis (1977), Linear Programming and Network Flow, John Wiley and Sons, New York.

2. Cho, D. Chinhyung, Ellis L. Johnson, Manfred Padberg, and M.R. Rao (1981) "On the Uncapacitated Plant Location Problem I: Valid Inequalities and Facets," Working paper series #81-18, Faculty of Business Administration, New York University, May.

3. Erickson, Ranel (1978), "Minimum Concave-Cost Single-Source Network Flows," Ph.D. dissertation, Stanford University.

4. Garey, Michael R. and David S. Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco.

5. Kennington, Jeff L. and Richard V. Helgason (1980) Algorithms for Network Programming, John Wiley and Sons, New York.

6. Khachian, L.G. (1979) "A Polynomial Algorithm in Linear Programming," Doklady, Akademii Nank,SSSR, 244.

7. Rardin, R. L. and Vi Choe (1979) "Tighter Relaxations of Fixed Charge Network Flow Problems," Industrial and Systems Engineering Report Series Number J-79-18, Georgia Institute of Technology, May.

8. Truemper, K. (1977) "Unimodular Matrices of Flow Problems with Additional Constraints," Networks, 7, 343-358.

9. Truemper, K. and Y. Soun (1979), "Minimal Forbidden Subgraphs of Unimodular Multicommodity Networks," Mathematics of Operations Research, 4, 379-389.

10. Vienott, A.F., Jr., and G. B. Dantzig (1968), "Integer Extreme Points," SIAM Review, 10, 371-372.

NOTICE OF RESEARCH PROJECT

SCIENCE INFORMATION EXCHANGE

SMITHSONIAN INSTITUTION
NATIONAL SCIENCE FOUNDATION

PROJECT SUMMARY

SIE PROJECT NO.
NSF AWARD NO.

FOR NSF USE ONLY			
DIRECTORATE/DIVISION	PROGRAM OR SECTION	PROPOSAL NO.	F.Y.

NAME OF INSTITUTION (INCLUDE BRANCH/CAMPUS AND SCHOOL OR DIVISION)

Georgia Institute of Technology
School of Industrial and Systems Engineering

ADDRESS (INCLUDE DEPARTMENT)

225 North Avenue, N.W.
Atlanta, GA 30332

PRINCIPAL INVESTIGATOR(S)

Robert G. Parker

TITLE OF PROJECT

Tight Relaxation Approaches to Fixed Change Problems on Graphs and Networks

TECHNICAL ABSTRACT (LIMIT TO 22 PICA OR 18 ELITE TYPEWRITTEN LINES)

Previous results by the authors have demonstrated the solvability of certain combinatorial problems which can be formulated in a fixed charge network flow format. In particular, problems defined on series-parallel graphs can be solved in polynomial time since their linear programming relaxation is perfect. This research is directed at extending these promising results to include the enlargement of the class of problems having perfect relaxations as well as the development of more direct, cominatorial procedures. The latter has already been accomplished (by the authors) for the Steiner tree problem. Also, we propose an investigation of the applicability of employing the existing relaxation methodology to non-series-parallel structures in order to obtain fast nonexact procedures having finite worst-case bounds on their performance.

The present document is a slight modification of an earlier proposal (ECS 8300533) submitted by R. L. Rardin and R. G. Parker to the National Science Foundation and which was favorably reviewed. The modification stems from two sources. The first is Dr. Rardin's acceptance of a new position at Purdue University and the second, a request by NSF to reduce the scope of the earlier proposal. Accordingly, the new proposed effort is reduced to 1 year and $57,644 ($26,745 at Georgia Tech and $30,899 at Purdue). The technical content in the earlier proposal remains in tact with both investigators working jointly on all phases of the research.

1. Proposal Folder
2. Program Suspense
3. Division of Grants & Contracts
4. Science Information Exchange
5. Principal Investigator
6. Off. of Govt. & Pub. Progs.

JOINT STATEMENT

Our proposal "Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks", (ECS 8300533) was submitted to the National Science Foundation (NSF) in fall 1982. (A copy of the technical section is attached.) It envisioned a 2 year program of research budgeted at $174,956 with Dr. Ronald L. Rardin as Principal Investigator and Dr. R. Gary Parker as Co-Principal Investigator -- both working at the Georgia Institute of Technology.

Since that time the proposal has been favorably reviewed by NSF. At the same time Dr. Rardin has accepted a new position as Professor of Industrial Engineering at Purdue University.

This statement accompanies two modifications of the earlier proposal. One provides support for Dr. Parker at Georgia Institute of Technology; the other for Dr. Rardin at Purdue. We intend the two proposals to together address the same technical content as the earlier one and request that they be evaluated together.

At the request of NSF, the total scope of the earlier proposed effort has been reduced to 1 year and $57,644 ($26,745 at Georgia Tech and $30,899 at Purdue). To accomodate this shorter time we have deleted all implementation of results as computer algorithms, emphasizing instead the extension of theory on which future algorithms will depend. We expect to pursue all theoretical aspects of the earlier proposed work -- including direct combinatorial algorithms, polyhedral properties, and nonexact algorithms. However, priority will be given to direct algorithms because they are necessary components of other developments.

Although working at different institutions, we expect to closely coordinate our activities and take joint part in all phases of the research. However, there will be some difference of emphasis.

Dr. Rardin will concentrate his research on polyhedral properties and expansion of the class of tractable problems. Dr. Parker will emphasize underlying combinatorial structure and nonexact algorithms. Travel funds provided in our respective budgets will provide for sufficient joint meetings at our campuses or during research conferences to effect required coordination.

Ronald L. Rardin, Ph.D.
Principal Investigator
Purdue University

R. Gary Parker, Ph.D.
Principal Investigator
Georgia Institute of Technology

E-24-631

NATIONAL SCIENCE FOUNDATION
Washington, D.C. 20550

FINAL PROJECT REPORT
NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I–PROJECT IDENTIFICATION INFORMATION

1. Institution and Address	2. NSF Program	3. NSF Award Number
Georgia Institute of Technology Atlanta, Georgia 30332	Systems Theory and Operations Research	ECS-8018954
	4. Award Period From 3/15/81 To 8/31/83	5. Cumulative Award Amount $49,951

6. Project Title

"Tight Relaxation Approaches to Fixed Charge Problems on Graphs and Networks"

PART II–SUMMARY OF COMPLETED PROJECT *(FOR PUBLIC USE)*

The principal focus of this research is on the development of effective solution strategies for dealing with hard combinatorial problems which can be formulated generally, as fixed charge flow problems on graphs and networks. Numerous examples fit this description among which are the Steiner tree problem, the minimum equivalent subgraph problem and the traveling salesman problem. In the earlier stages of the grant period, effort was concentrated on more global issues regarding non-standard linear programming relaxation for general fixed charge formulations. Here, the notion of progressive linear programming relaxation involving a disaggregation strategy was developed and tested. Following this, research shifted to direct, combinatorial approaches for problems of the aforementioned type. Important in this regard is that when instances of such problems (and others) are defined on the class of graphs known as series-parallel, exact solutions can be obtained in polynomial time. Exploiting this, numerous algorithms were developed for weighted versions of various problems thus subsuming cardinality cases as well. The research concluded with some treatise of heuristic analysis in non-series-parallel settings, specific emphasis being placed on the construction of (unimprovable) performance guarantees. Here, such a result was established for the bottleneck traveling salesman problem using notions pertaining to the square of biconnected graphs.

PART III–TECHNICAL INFORMATION *(FOR PROGRAM MANAGEMENT USES)*

ITEM *(Check appropriate blocks)*	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (✓)	Approx. Date
a. Abstracts of Theses		X			
b. Publication Citations		X			
c. Data on Scientific Collaborators		X			
d. Information on Inventions	X				
e. Technical Description of Project and Results		X			
f. Other *(specify)*					

1. Principal Investigator/Project Director Name *(Typed)*	3. Principal Investigator/Project Director Signature	4. Date
R. Gary Parker		1/9/84

Form 98A (5-78) Supersedes All Previous Editions

Form Approved OMB No. 99R0013

ATTACHMENT 1

Thesis Abstract

Thesis summary for: "Progressive Disaggregation for Fixed Charge Network Flow Problems", by Oscar Adaniya

Advisor: Dr. R. L. Rardin

SUMMARY

Fixed charge network flow problems model network design and location settings by allowing both fixed and variable charges for arc flow. Recent research has shown that very close approximations to mixed-integer solutions for each problem can be obtained from massive linear programs wherein flows are artificially disaggregated into separate components for each origin-destination pair. This paper develops the strategy of a progressive disaggregation algorithm employing the latter linear programming relaxation. However, flows are initially undisaggregated. As computation proceeds, supply and demand subsets are further and further partitioned to tighten the relaxation as required without incurring the computational burden of a complete disaggregation into supply-demand pairs.

ATTACHMENT 2

Publication Citations

Rardin, R. L., R. G. Parker, and D. Wagner (1983), "Definitions, Properties and Algorithms for Detecting Series-Parallel Graphs," ISyE Report Series, Georgia Inst. Tech., Atlanta, GA.

Rardin, R. L., R. G. Parker and M. B. Richey (1982), "A Polynomial-Time Algorithm for the Steiner Tree Problem on Graphs," ISyE Report Series J-82-5, Georgia Inst. Tech., Atlanta, GA.

Richey, M. B., R. G. Parker and R. L. Rardin (1983), "A Solvable Case of the Minimum Weight Equivalent Subgraph Problem," ISyE Report Series, Georgia Inst. Tech., Atlanta, GA. (Submitted to Networks)

Richey, M. B., R. G. Parker and R. L. Rardin (1982), "On a Class of Graphs Having at Most One Hamiltonian Cycle," ISyE Report Series, J-82-11, Georgia Inst. Tech., Atlanta, GA.

Parker, R. G. and R. L. Rardin (1983), "Guaranteed Performance Heuristics for the Bottleneck Traveling Salesman Problem," Operations Research Letters, (to appear).

Rardin, R. L. and R. G. Parker (1983), "On Producing a Hamiltonian Cycle in the Square of a Biconnected Graph: An Algorithm and Its Use," ISyE Report Series, J-83-01, Georgia Inst. Tech., Atlanta, GA. (Submitted to Math. of O.R.)

Rardin, R. L. and O. Adaniya (1982), "Development of a Progressive Disaggregation Algorithm for Fixed Charge Network Flow Problems," ISyE Report Series J-82-4, Georgia Inst. Tech., Atlanta, GA.

Rardin, R. L. (1982), "Tight Relaxations of Fixed Charge Network Flow Problems," ISyE Report Series, J-82-3, Georgia Inst. Tech., Atlanta, GA.

ATTACHMENT 3

Collaborators

R. L. Rardin, Professor

School of Industrial Engineering

Purdue University

W. Lafayette, IN 47907

Oscar Adaniya, Assistant Professor

Industrial Engineering Department

University of Miami

Coral Gables, Florida 33124

(Dr. Adaniya was a graduate student at the time his effort was realized on the project.)

M. B. Richey, Graduate (Ph.D.) Student

School of Industrial and Systems Engineering

Georgia Institute of Technology

Atlanta, Georgia 30332

ATTACHMENT 4

Technical Summary

TECHNICAL SUMMARY

A vast number of important optimization problems in areas such as distribution, communication, transportation and facilities location can be viewed as flow problems on graphs or networks, where flow is permitted in certain arcs/edges, (i,j) only when a fixed charge, f_{ij}, is paid. Flow originates at known supply points in the graphs, and distributes to demand points of known requirements while satisfying some balance of flow considerations at intermediate transshipment points. Flow may be in a single or in multiple commodities, and flow capacities may or may not be present.

When the graph is directed, such fixed charge network flow problems may be formulated as

(NP)

$$\min \sum_{p\in P} \sum_{(i,j)\in E} v_{ij}^p x_{ij}^p + \sum_{(i,j)\in E} f_{ij} y_{ij} \qquad (1)$$

$$\text{s.t.} \sum_{(i,\beta)\in E} x_{i\beta}^p = d_\beta^p \qquad \text{for all } p\in P,\ \beta\in D^p \qquad (2)$$

$$\sum_{(\alpha,j)\in E} x_{\alpha j}^p \le s_\alpha^p \qquad \text{for all } p\in P,\ \alpha\in S^p \qquad (3)$$

$$\sum_{(\ell,j)\in E} x_{\ell j}^p - \sum_{(i,\ell)\in E} x_{i\ell}^p = 0 \qquad \text{for all } p\in P,\ \ell\in T^p \qquad (4)$$

$$\Big(\sum_{p\in P} x_{ij}^p\Big)/u_{ij} \le y_{ij} \qquad \text{for all } (k,j)\in E \qquad (5)$$

$$x_{ij}^{p} \geq 0 \qquad \text{for all } (i,j)\in E,\ p\in P \qquad (6)$$

$$1 \geq y_{ij} \geq 0 \qquad \text{for all } (i,j)\in E \qquad (7)$$

$$y_{ij} \text{ integer} \qquad \text{for all } (i,j)\in E \qquad (8)$$

Here E is the arc set of the specified structure; P is the set of commodities; x_{ij}^{p} is the flow of commodity p from i to j; S^{p}, D^{p} and T^{p} are the supply point, demand point, and transshipment point subsets for commodity p, respectively; s_{α}^{p} is the commodity p supply at point α; d_{β}^{p} is the commodity p demand at point β; and u_{ij} is a flow capacity for arc (i,j). Costs in (1) include a variable (per unit flow) cost v_{ij}^{p} and a fixed charge f_{ij} "switched on" by the 0-1 variable y_{ij} whenever any $x_{ij}^{p} > 0$. We assume throughout that all f_{ij} and v_{ij} are nonnegative although the latter requirement can be relaxed in some cases.

We term problems of the form (NP) uncapacitated if the optimal solution set would not change when all s_{α}^{p} in (2) and u_{ij} in (5) were replaced by a large constant, M (say $\sum_{p\in P}\sum_{\beta\in D^{p}} d_{\beta}^{p}$). The problems are weakly capacitated if only the u_{ij} may be replaced by such an M, and capacitated if both the s_{α}^{p} and the u_{ij} restrict solutions.

One particularly straightforward example of a problem which takes on form (NP) is the classic warehouse location problem. However, numerous other well-known discrete optiization problems can be easily, although sometimes less obviously, cast in the (NP) form. Among these is the

Steiner tree problem, the minimum weight equivalent subgraph problem, and the traveing salesman problem.

Unfortunately, most interesting models of the (NP) form are difficult integer and combinatorial programming problems. In fact, many can be shown to belong to the notorious class NP-Complete. Thus, almost all research on practical algorithms for such problems centers on either enumerative, branch-and-bound schemes or approximate procedures yielding feasible, but not provably optimal solutions. Such techniques rely, in turn, on relaxations of the original problem, i.e., problems with feasible solution sets including that of (NP) and cost or objective functions underestimating (1). Such relaxations may, of course, be much easier to solve than the original problem. Accordingly, if they are sufficiently tight (i.e., they closely approximate the original problem), relaxations can provide useful lower bounds for branch-and-bound algorithms and serve as the core of heuristic procedures for constructing good feasible solutions to (NP).

The majority of solution procedures draw on the linear programming relaxations obtained when the integrality requirements (8) are discarded. Substantial research has been done, accordingly, relative to the formulation of tigher relaxations. Rardin and Choe (1979) pursued this line of investigation. There, flows x_{ij}^{p} for each true commodity $p \in P$ are disaggregated into components $x_{ij}[\alpha,\beta]$ distinguished by the suply point α at which the flow originated and the demand point β to which it is destined. Viewing each α,β,p combination as a separate comodity yields the formulation:

$$\min \sum_{p\in P} \sum_{(i,j)\in E} v_{ij}^{p} \left[\sum_{\alpha\in S^p} \sum_{\beta\in D^p} x_{ij}^{p}[\alpha,\beta] \right] + \sum_{(i,j)\in E} f_{ij}\, y_{ij} \qquad (10)$$

$$\text{s.t.} \quad \sum_{\alpha\in S^p} \sum_{(i,\beta)\in E} x_{i\beta}^{p}[\alpha,\beta] = d_{\beta}^{p} \qquad \text{for all } p\in P,\ \beta\in D^p \qquad (11)$$

$$\sum_{\beta\in D^p} \sum_{(\alpha,j)\in E} x_{\alpha j}[\alpha,\beta] \leq s_{\alpha}^{p} \qquad \text{for all } p\in P,\ \alpha\in S^p \qquad (12)$$

(MC)

$$\sum_{(\ell,j)\in E} x_{\ell j}^{p}[\alpha,\beta] - \sum_{(i,\ell)\in E} x_{i\ell}^{p}[\alpha,\beta] = 0 \qquad \text{for all } p\in P,\ \alpha\in S^p,\ \beta\in D^p,\ \ell\in T^p \qquad (13)$$

$$(1/u_{ij}) \sum_{p\in P} \sum_{\alpha\in S^p} \sum_{\beta\in D^p} x_{ij}^{p}[\alpha,\beta] \leq y_{ij} \qquad \text{for all } (i,j)\in E \qquad (14)$$

$$x_{ij}^{p}[\alpha,\beta] \geq 0 \qquad \text{for all } (i,j)\in E,\ p\in P,\ \alpha\in S^p,\ \beta\in D^p \qquad (15)$$

$$1 \geq y_{ij} \geq 0 \qquad \text{for all } (i,j)\in E \qquad (16)$$

$$y_{ij} \text{ integer} \qquad \text{for all } i,j)\in E \qquad (17)$$

$$\frac{x_{ij}^{p}[\alpha,\beta]}{\min\{s_{\alpha}^{p}, d_{\beta}^{p}\}} \leq y_{ij} \qquad \text{for all } (i,j)\in E,\ p\in P,\ \alpha\in S^p,\ \beta\in D^p \qquad (18)$$

Of course, the linear programming relaxation of (MC) need not yield an (MC) optimum, but preliminary testing in Rardin and Choe (1979) indicated the relaxed formulation could provide a very tight relaxation for at least the uncapacitated and weakly capacitated cases.

Now, constraints (18) of the tight (MC) formulation requires y_{ij} to equal or exceed the fraction of any supply s_α^p or demand d_β^p flowing as $x_{ij}^p[\alpha,\beta]$. However, the formulation can be sharpened if (18) is replaced by

$$\frac{\sum_{\beta \in \mathcal{D}^p} x_{ij}^p[\alpha,\beta]}{\min\left\{s_\alpha^p, \sum_{\beta \in \mathcal{D}^p} d_\beta^p\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E, \; p \in P, \; \alpha \in S^p \qquad (19)$$

$$\frac{\sum_{\alpha \in S^p} x_{ij}^p[\alpha,\beta]}{\min\left\{\sum_{\alpha \in S^p} s_\alpha^p, d_\beta^p\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \; p \in P, \; \beta \in \mathcal{D}^p \qquad (20)$$

These new constraints treat supplies and demands separately, summing flows in one of the two dimensions.

Denote by (MC') the version of formulation (MC) using (19) and (20) in place of (18). It is easy to show that the (MC') formulation dominates (MC) in the sense that every solution feasible in (MC') is feasible for (MC). However, (MC') is also more compact. For every commodity p there is one constraint (18) for each $(\alpha,\beta) \in S^p, \mathcal{D}^p$, i.e., $|S^p| \cdot |P^p|$ in all. The stronger (MC') formulation is achieved with $|S^p| +$

$|\mathcal{D}^p|$ constraints (19) and (20).

Of course, linear relaxation of tight formulations (MC) and (MC') are enormous linear programs requiring specially structured algorithms. For a case with 5 commodities, 350 arcs, 10 supplies, 50 demands and 130 transshipment vertices, (MC') has over 425 thousand main constraints and 875 thousand variables.

The early (Rardin and Choe (1979)) paper detailed a Lagrangean relaxation strategy for the (MC') linear relaxation of the uncapacitated and weakly capacitated cases. For those cases, constraints (14) may be deleted. Dualization of constraints (19) and (20) with Lagrange multipliers leaves a separate flow problem for each α,β,p combination. The latter can be solved by a series of shortest path calculations followed by a transportation problem. Search over dual multipliers for constraints (19) and (20) leads to a relaxation solution.

The formulation (NP) and (MC) (or (MC')) may be viewed as endpoints or a disaggregation continuum. Form (NP) treats all flows of commodity p as a single unit; (MC') disaggregates flows for each p into separate artificial commodities for origin-destination pairs. Clearly, there are intermediate possibilities wherein commodity p flows are treated in groupings, say (A_k^p, B_k^p) with $A_k^p \subset S^p$, $B_k^p \subset \mathcal{D}^p$.

The concept of <u>progressive disaggregation</u> suggested in the funded proposal sought to exploit these intermediate possibilities in order to speed computation. Flows might begin in fully aggregated form (NP). As computation proceeds, and dual variables are better estimated, smaller supply and demand groups could be attempted in order to improve the quality of the relaxation.

A major part of the present resssearch effort has been devoted to developing such a progressive approach for implementing the Lagrangean dual algorithm on uncapacitated and weakly capacitated cases. Various issues are analyzed among which are

- Which are the most desirable forms of supply group-demand group commodizations?
- How can an initial list of artificial commodities be generated?
- How should supply or demand groups be selected for partition as disaggregation proceeds?
- When should additional disaggregation be involved?
- How can dual variables for new commodity groups be effectively initialized from ones for groups they replace?

Tables 1 and 2 summarize computational results. Twelve different uncapacitated and weakly capacitated fixed charge network flow problems were randomly generated with all combinations of three types of capacitization (uncapacitated, weakly capacitated with relatively loose capacity, weakly capacitated with tight capacity), two problem sizes (175 and 350 arcs), and two levels of fixed charge contribution to cost (30-40% versus 60-70%). All problems are sparse. The problems were solved with 6 algorithmic strategies (only 3 are applicable to uncapacitated cases) involving all combinations of initial supply disaggregation (no initial disaggregation, full initial disaggregation) and initial disaggregation (no initial disaggregation, selected initial group formulation, full initial disaggregation) alternatives. Since the goal was comparison of strategies, all cases were terminated when a primal solution and a lower bound were known to differ by at most 25%.

Table 1. Medium Problem Results
(175 arcs, 5 supplies, 25 demands, 75 nodes)

Initial Supply Disaggr.	Initial Demand Disaggr.	Uncapacitated		Weakly Capacitated (Loose)		Weakly Capacitated (Tight)	
		Moderate Fixed	High Fixed	Moderate Fixed	High Fixed	Moderate Fixed	High Fixed
None	None	3	3	7	7	7	8
	Selected	5	8	13	11	18	12
	Full	7	12	32	34	32	29
Full	None			19	6	9	11
	Selected			12	12	13	19
	Full			22	27	24	18

Table 2. Large Problem Results[1,2/]
(350 arcs, 10 supplies, 50 demands, 150 nodes)

Initial Supply Disaggr.	Initial Demand Disaggr.	Uncapacitated		Weakly Capacitated (Loose)		Weakly Capacitated (Tight)	
		Moderate Fixed	High Fixed	Moderate Fixed	High Fixed	Moderate Fixed	High Fixed
None	None	3	8	60	64	60	109
	Selected	8	27	>180	>180	122	>180
	Full	14	40	>180	>180	>180	>180
Full	None			>180	>180	174	>180
	Selected			93	>180	179	>180
	Full			>180	>180	>180	>180

[1/] All times in Univac 1100/81 minutes. Typically 8-10% was CPU with the residual being disk operations.

[2/] Times reflect solution to provable 25% optimality.

Results in the tables clearly demonstrate the merit of the progressive disaggregation approach. All progressive strategies produced better results than the "brute force" approach which fully disaggregates supplies and/or demands before computation begins. The best progressive strategy--starting with no disaggregation of supplies or demands--was 2 to 4 times more efficient than the complete disaggregation approach and on some alrge problems, the only method to yield results within the time limit.

Of course we would like to have a relaxation (say (MC')) which under fairly mild restrictions was perfect in the sense that its solution was integer-optimal. However, even if such a development was in hand,appeal to the polynomial solvability of linear programs vis-a-vis the ellipsoid algorithm would not, presently, have great practical value. Rather more direct, combinatorial approaches would be sought.

Accordingly, this research also examines the development of such efficient procedures for problems defined on a restricted class of graphs known as series-parallel. Such research has appeared elsewhere in various forms. Notable in this regard is the work reported in Takamizawa, Nishizeki and Saito (1982).

A graph is series-parallel if and only if it contains no subgraph homeomorphic from K_4 (the complete graph on four vertices). Other specifications of series-parallel also exist, their equivalence being shown in the unifying paper by Rardin, Parker and Wagner (1982).

Polynomial-time algorithms have been given for weighted versions of the Steiner tree problem and the minimum equivalent subgraph problem. these procedures are detailed in other aattachments. Also included is the problem of deciding hamiltonicity in series-parallel graphs. Here,

we prove that such graphs have at most one such cycle and we characterize those that are hamiltonian.

Finally, the notion of nonexact analysis is considered relative to non-series-parallel structures. Our interest is confined to those procedures which are not improvable by alternative, polynomial schemes in terms of their performance guarantees.

Our principle finding along these lines is somewhat negative. For the so-called bottleneck traveling salesman problem (BTSP), we were able to produce a nonexact procedure having worse-case bound of two which is realizable and not improvable by any polynomial alternative unless and are equivalent. The stated algorithm is based on the notion of squares of biconnected graphs. Such graphs (squares) are known to be hamiltonian. In the attachments, we give an algorithm for finding such a cycle as well as the nonexact analysis for the BTSP.

REFERENCES

Rardin, R. L. and Ui Choe (1979), "Tighter Relaxations of Fixed Charge Network Flow Problems," ISyE Report Series J-79-18, Georgia FInst. Tech., Atlanta, GA.

Rardin, R. L., R. G. Parker, and D. Wagner (1983), "Definitions, Properties and Algorithms for Detecting Series-Parallel Graphs," ISyE Report Series, Georgia Inst. Tech., Atlanta, GA.

Takamizawa, K., T. Nishizeki, and N. Saito (1982), "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs," JCAM, 29, 3, pp. 623-641.

ATTACHMENT 5

Technical Reports

DEFINITIONS, PROPERTIES AND ALGORITHMS FOR DETECTING SERIES-PARALLEL GRAPHS

by

R. L. Rardin*, R. Gary Parker† and D. K. Wagner*

*Department of Industrial Engineering
Purdue University
W. Lafayette, IN 47907

†School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS-8018954

ABSTRACT

Series-parallel graphs form an important subset of planar graphs defined in terms of arrangements of edges and subgraphs in a manner corresponding to series and parallel connection in electrical networks. In this paper we review a host of alternative definitions of such graphs and show that if properly specified, all definitions are equivalent. We also exhibit a linear-time algorithm for checking such properties.

1. Introduction

Series-parallel graphs are an important subset of planar graphs defined in terms of arrangements of edges or subgraphs in a fashion corresponding to the intuitive notion of series and parallel connection in electrical networks. A classic study of such graphs is Duffin (1965), and recent results are contained in Takamizawa, Nishizeki, and Saito (1982), and Valdes, Tarjan, and Lawler (1982), Rardin, Parker, and Wang (1982), and Rardin, Parker, and Rickey (1982).

Unfortunately, there are some disparaties regarding definitions of such graphs. The purpose of this note is to verify that, when carefully specified, the concepts behind all these definitions can be made to conform, i.e. all such characterizations are equivalent. We also exhibit a linear-time algorithm for implementing one of the definitions and indicate how it implicitly tests several others.

Neither the algorithm nor any of our characterizations is entirely, or even mostly new. Rather, our objective is only to make precise, matters which are more often hinted at than exposited in the extensive literature of series-parallel graphs.

2. Main Definition

We consider an undirected, loopless[1/] multigraph G with finite vertex set V and finite edge set E, such that V contains no isolated vertices. We shall say that such a G is series-parallel if and only if some sequence of applications of the following three reductions converts G into a disjoint collection of edges.

1/ Results are easily extended to encompass self loops by introducing an artificial, degree-2 vertex into each loop before applying our results.

- series reduction: replace any degree-2 vertex j and its incident edges e = (i,j), f = (j,k) such that $i \neq k$, by new edge g = (i,k).
- parallel reduction: replace any two edges e = (i,j) and f = (i,j) joining the same vertices by a new edge g = (i,j).
- jackknife reduction: replace any degree-1 vertex i, its incident edge e = (i,j), and any other edge f = (j,k) meeting vertex j by a new edge g = (j,k).

Figure 1 illustrates the three types of reductions, and Figure 2 shows a (not unique) sequence reducing a given graph to a single edge. Observe that the three reductions are well-defined in that any suitable sequence will produce the correct conclusion regarding a given input graph.

Series and parallel reductions are well known and common to all definitions. However, confusion arises when G is not connected or is not biconnected. "Two-terminal series parallel" in Takamizawa et al (1982), "edge series-parallel" in Valdez et al (1982) and "closed graphs" of Duffin (1965) are constrictions to distinguish various cases. We introduce jackknife reductions to encompass non-biconnected situations such as that of Figure 3. It is easy to verify that all three of the above reductions are needed to reduce the graph of Figure 3 to a single edge.

3. Forbidden Homeomorphic Subgraphs

A subgraph H of G is said to be homeomorphic from a graph Q if and only if some sequence of applications of series reduction to H produces a graph isomorphic to Q. Series-parallel graphs can be characterized in terms of forbidden homeomorphic subgraphs.

Theorem 1: Series-Parallel Graphs and K_4. A loopless, undirected, multigraph G with no isolated vertices is series-parallel if and only if G contains no subgraph homeomorphic from K_4 (the complete graph on 4 vertices).

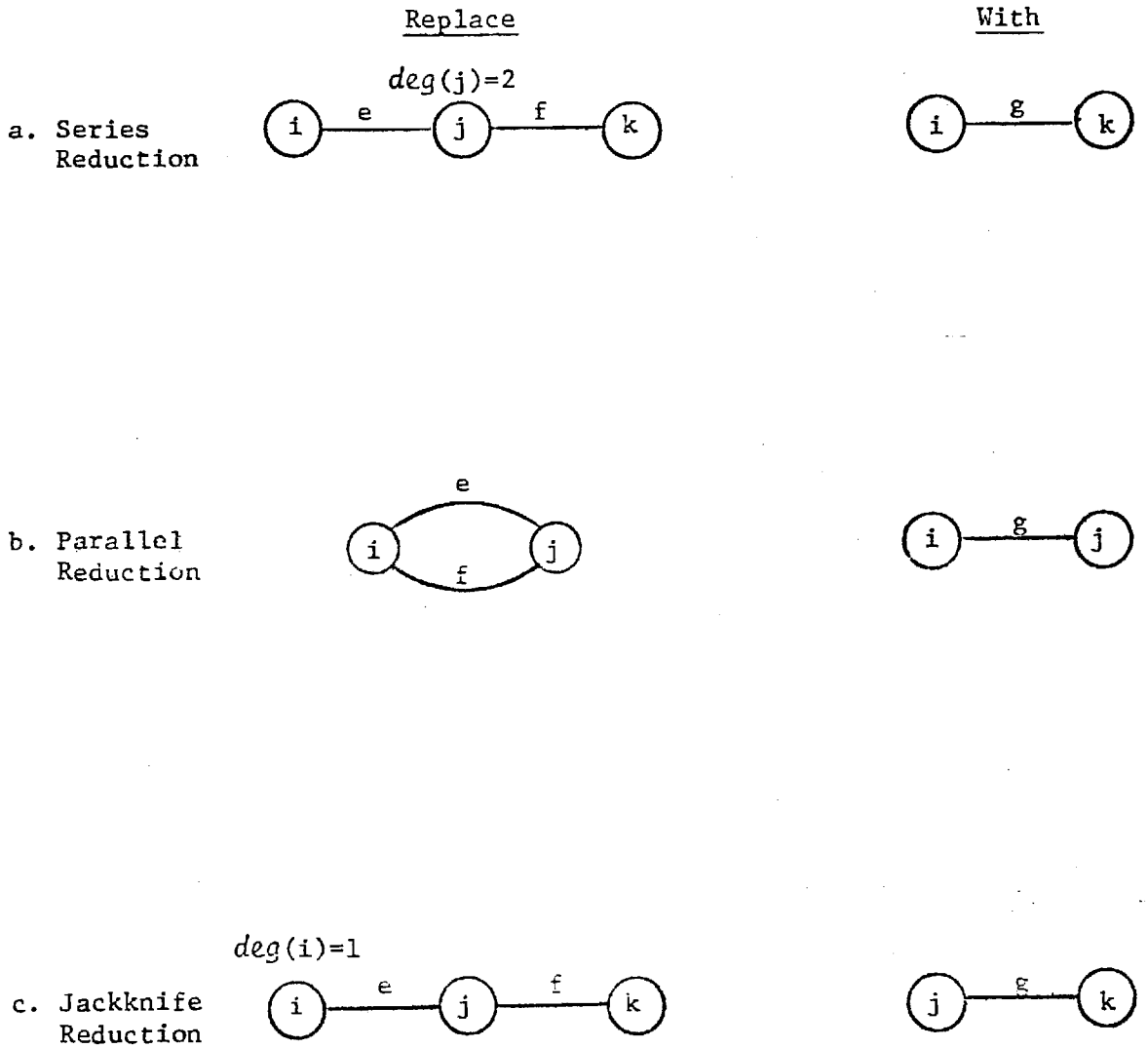


Figure 1: Defining Reductions of Series-Parallel Graphs

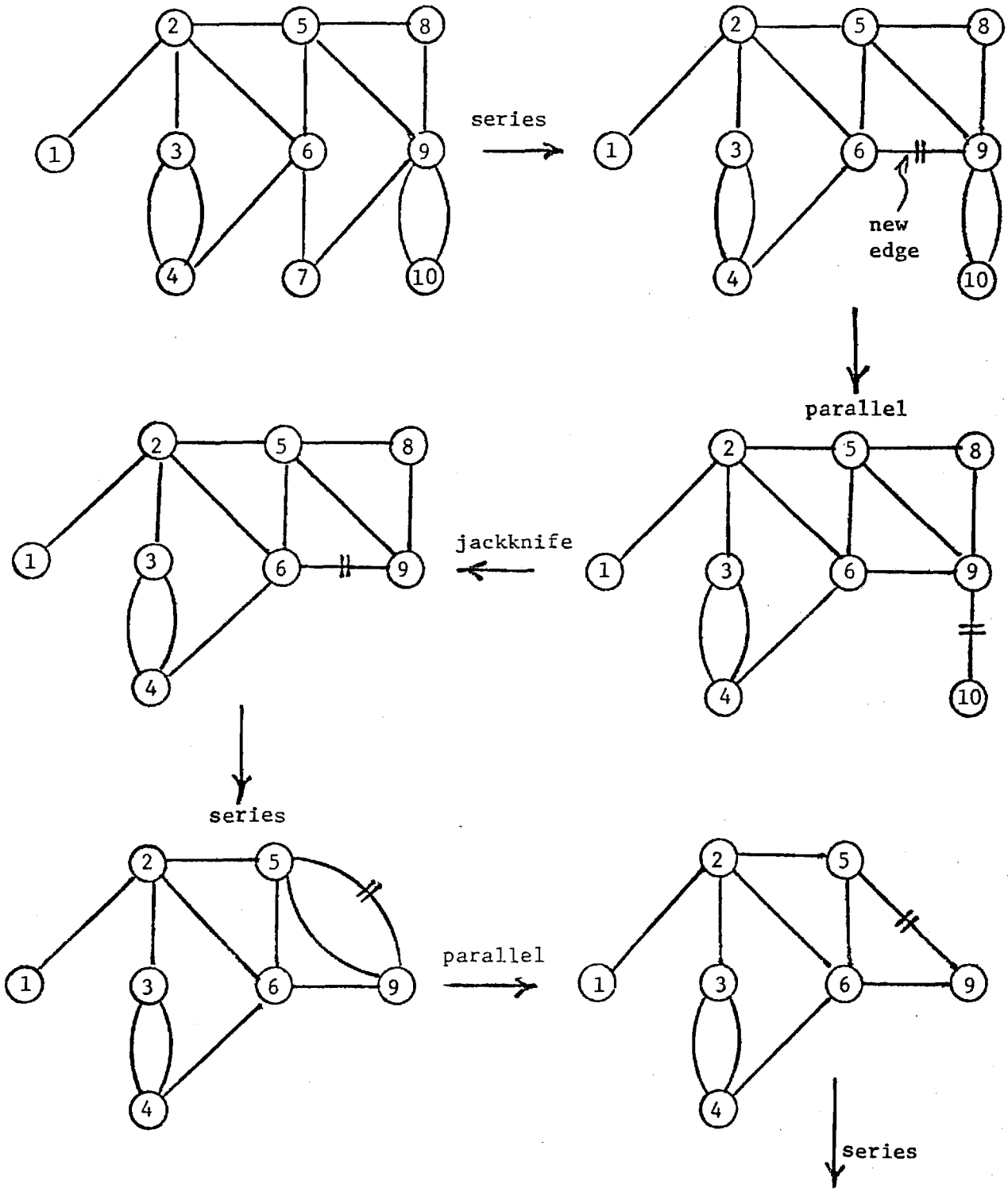


Figure 2: A Sequence of Reductions Showing a Given Graph is Series Parallel

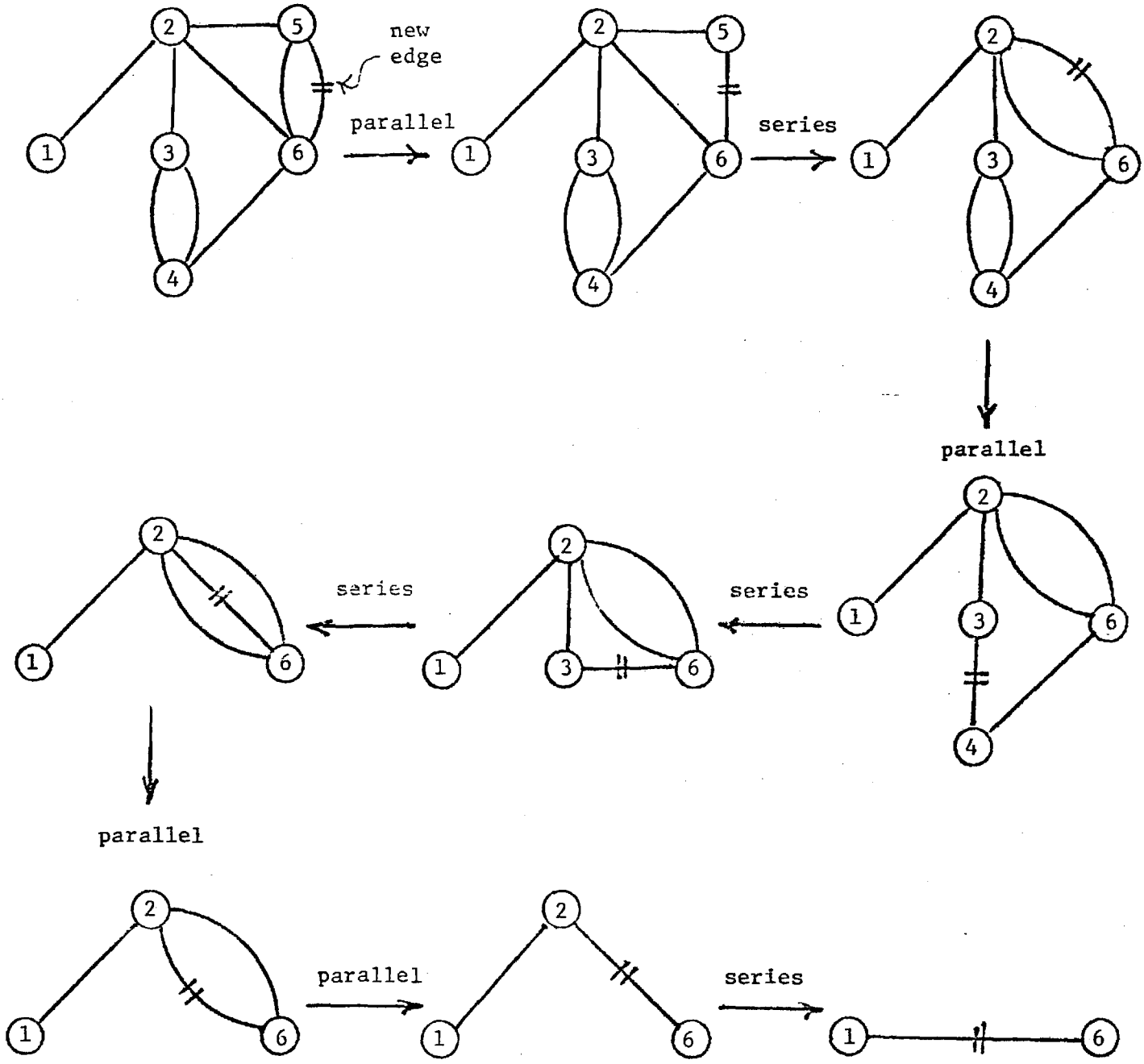


Figure 2 (continued)

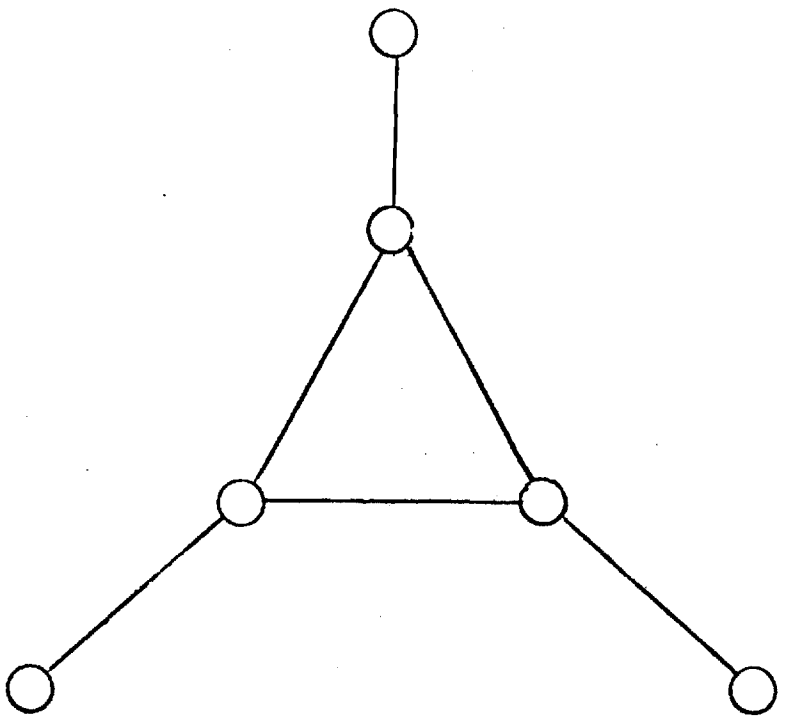


Figure 3. A Graph Requiring Series, Parallel and Jackknife Reductions to be Reduced to an Edge.

Proof: First observe that we need consider only connected graphs G. If any subgraph homeomorphic from K_4 exists, one will certainly lie in a single component of G. Conversely, series, parallel and jackknife reductions neither combine nor create disjoint components, so that G is series-parallel if and only if every connected component can be reduced to a single edge.

Assume G is connected, and let H be a subgraph of G homeomorphic from K_4. Any G' obtained from G by a single series, parallel or jackknife reduction must also contain a subgraph H' homeomorphic to K_4. This is true because if all reduced edges belong to $E(G)\backslash E(H)$ then $H' = H$ (note that we symbolize the edge and vertex sets of a graph G by $E(G)$ and $V(G)$ respectively). If not, H' is either the product of a series reduction of H, or the result of replacing an edge e or f of H by a new edge g joining the same vertices.

Applying this observation at each step of series, parallel, or jackknife reduction, we see that every sequence of such reductions must terminate before or upon reducing G to a graph isomorphic to K_4. Since K_4 admits no further reduction, we can conclude that if the original G contains a subgraph homeomorphic from K_4, no sequence of reductions will reduce G to a single edge, i.e. G is not series-parallel.

For the converse, we must show that if G is not series-parallel, i.e. reduction terminates before G has been reduced to an edge, then G contains a subgraph homeomorphic from K_4. The algorithm of Section 4 and Lemma 3 will demonstrate constructively that if reduction terminates on a reduced form G' of G, then G' contains a subgraph H' homeomorphic from K_4. By the same arguement as above, it follows that G must have contained a corresponding subgraph H that has been reduced to H' in G'.

We call a vertex, c, a cut-vertex of a graph G if c separates G into subgraphs G_1 and G_2 with nonempty edge set, and such that $G = E(G_1) \cup E(G_2)$ and $V(G_1) \cap V(G_2) = \{c\}$. A cut-pair is a pair of vertices c and d of G that separates G into subgraphs G_1 and G_2 with nonempty edge sets such that $G = E(G_1) \cup E(G_2)$, $V(G_1) \cap V(G_2) = \{c,d\}$. A graph is triconnected if it is connected, contains at least 4 vertices, and possesses no cut-vertex and no cut-pair that separates G into G_1 and G_2 with both G_1 and G_2 having at least three vertices.

The graph K_4 of Theorem 1 is triconnected. We can apply Theorem 1 to see that series-parallel graphs forbid any triconnected homeomorphic subgraphs.

Corollary 2: Triconnectivity and Series-Parallel Graphs. A loopless, undirected graph G with no isolated vertices is series-parallel if and only if it contains no subgraph homeomorphic from a triconnected graph.

Proof: If G is not series-parallel, Theorem 1 shows G contains a subgraph H homeomorphic from K_4. Since K_4 is triconnected, G does indeed have a subgraph homeomorphic from a triconnected graph.

For the converse suppose G contains a subgraph H homeomorphic from a triconnected graph Q. Then G must also contain such a subgraph $\overline{H}$ homeomorphic from a triconnected graph $\overline{Q}$ having no multiedges. Removal of edges in $Q_0 \triangleq Q$ would either produce the needed $\overline{Q}$ or generate a graph with degree-2 vertices which is itself homeomorphic from some triconnected Q_1. If Q_1 has multiedges, we repeat the process.

Let $\overline{Q}$ be a triconnected graph with no multiedges from which subgraph $\overline{H}$ of G is homeomorphic. Clearly the minimum degree in $\overline{Q}$ is three;

neighbors of degree-1 and degree-2 vertices form cut vertices and cut-pairs respectively. But then neither series, parallel nor jackknife reduction of $\overline{Q}$ is possible. By Theorem 1, $\overline{Q}$ contains a subgraph homeomorphic from K_4. But since $\overline{H}$ is homeomorphic from $\overline{Q}$ amd $\overline{Q}$ is homeomorphic from K_4, we see $\overline{H}$ is homeomorphic from K_4. By Theorem 1, G is not series-parallel.

4. A Linear Time Algorithm

One version of the problem of finding a linear time algorithm for detecting series-parallel graphs is posed as an exercise in Aho, Hopcroft and Ullman (1974). Although no solution is given, Valdes et al (1982) provide one for graphs that can be fully reduced by just series and parallel reduction, and they indicate Valdes (1978) contains another. Takamizawa et al (1982) reference a Japanese-language publication, Nishizeki et al (1976) in asserting linear time testability of their forms of series-parallel graphs.

We shall show in Section 6 that none of these papers seems to deal with the full family of graphs we term series-parallel. Thus we shall present here our own linear time scheme. The algorithm is strongly based on Hopcroft and Tarjan's (1973) approach to the closely related problem of identifying triconnected components of a graph. As in the proof of Theorem 1, an algorithm to either reduce a given graph G satisfying the hypothesis of the theorem to a collection of disjoint edges, or demonstrate that G contains a subgraph homeomorphic to K_4 needs only to deal with connected components, one at a time. Thus we shall

take G to be a connected multigraph encoded by a list of pairs $\{u[e],v[e]: e \in E\}$ recording the two vertices joined by each e, and a set of star lists $\{s[v,i]: i=1,2,\ldots,d[v]\}$ showing indices of the d[v] edges joining each vertex $v \in V$. For simplicity of presentation we will also assume G contains a degree-1 vertex r. Obviously, if no such vertex existed, we could augment G with an artifical edge joining an artificial vertex without impacting whether or not G contained a subgraph homeomorphic from K_4.

The algorithm proceeds in up to four stages. The first stage searches G from r in depth first sequence, labeling vertices and edges for sorting at Stage II. Vertices are labeled with $\ell[v] \triangleq$ the depth of vertex v in the search, and $t[v] \triangleq$ the edge through which vertex v is first visited. When the search is completed, edges of $\{t[v] : v \in V\}$ yield a spanning tree of G with nontree edges forming backedges, i.e. $(u,v) \in E$ such that if $\ell[u] > \ell[v]$, v lies along the tree path from u to the root vertex r. Similarly, if $\ell[v] > \ell[u]$, u lies along the tree path from v to r. (See for example Aho, Hopcroft and Tarjan (1974), p. 178 for verification of this property of depth-first search.)

All vertices deeper in the tree than a given tree edge, t, are called descendents of t, including the v such that t = t[v]. Back edges have no descendents.

The edge labels b[e] that we compute reflect the minimum depth $\ell[w]$ of a vertex w reachable through a path beginning with e and using only decendents of e and their backedges. Specifically, for e = (u,v), $\ell[u] < \ell[v]$

$$b[e] \triangleq \begin{cases} \ell[u] \text{ if } e \text{ is a backedge} \\ \min\{\ell[v], \min\{\ell[w]: (x,w) \text{ is a backedge, } x \text{ is a descendent of } e\}\} \\ \quad \text{if } e \text{ is a tree edge} \end{cases}$$

Stage II of the algorithm performs a radix sort to rearrange star edge numbers $e \in \{s[v,i]\}$ at each vertex in decreasing $b[e]$ order. All edges, e, are linked into one of the $|V|$ chains according to their $b[e]$ labels. Chains are then unloaded in reverse depth order to create new reordered stars $\{s[v,i]: i=1,2,\ldots,n[v]\}$ containing only the $n[v] \leq d[v]$ edges searched from v at Stage I. The effect is that the depth first search of Stage III will come last to portions of the tree with backedges reaching nearest the root. Figure 4 shows both the Stage I and Stage III sequence of first visits to edges in the example of Figure 2.

The Third Stage of the algorithm actually performs series, parallel and jackknife reductions until either G has been reduced to a single edge or we are able to conclude G contains a subgraph homeomorphic from K_4. G is searched in depth first fashion from the same root, thus building the same tree as in Stage I. However, this time stars have been sorted by Stage II.

Tree edges leading to degree-1 vertices are immediately jackknife-reduced with their predecessors in the tree. Whenever a degree-2 vertex is encountered in the search, it is immediately series-reduced. Parallel reductions are detected when existing or created backedges duplicate either the tree edge t[v] or an already passed over back edge from v. If a back edge is encountered that is parallel to neither t[v] nor the passed over back edge, we are able to conclude G contains a subgraph homeomorphic from K_4.

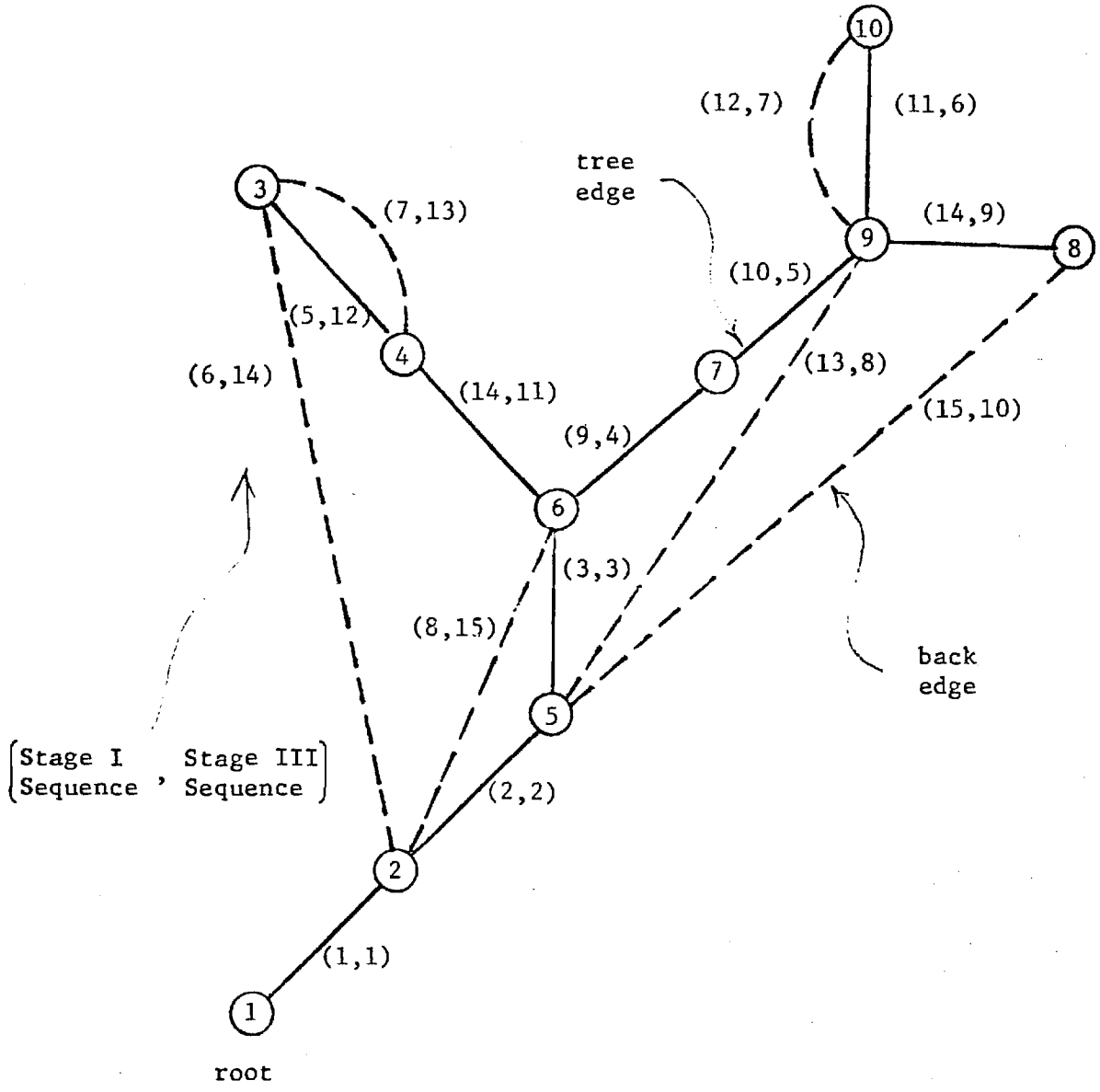


Figure 4: Depth First Search of
Example in Figure 2

As reductions develop, new edges are added to E. We dynamically update star values s[v,i] and free pointers t[v] to the new numbers, and record reductions in a binary tree. Specifically, we save

(u[e], v[e], p[e], c[e,1], c[e,2])

for each edge e=1,2,... where u[e] and v[e] are the two vertices joined by e, p[e] is the parent of e in the binary tree (the edge into which e is later reduced if any), and c[e,1], c[e,2] are the two edges from which e was created. Figure 5 shows the tree implied by reductions of Figure 2. The binary tree obtained could, itself, later be (linear time) searched in depth-first order to construct any original graph entity reduction has proved to be of interest.

If G is series-parallel, processing stops after Stage III returns to the root vertex r. However, if Stage III terminates with the conclusion that G contains a subgraph homeomorphic from K_4, Stage IV is applied to exhibit such a forbidden subgraph of the current G.[1/] The subgraph has the form shown in Figure 6. The single additional path needed to complete the homeomorph is identified by a partial continuation of the depth first search aborted in Stage III. The explicit statement of the algorithm can now be given.

Stage I: Edge Depth Labels

Step 0: Initialization. Tag all vertices as unvisited by setting

$$\ell[k] \leftarrow -1 \quad \text{for all } k \in V$$

1/ Backtracking in the binary tree might be required to exhibit a corresponding homeomorph from K_4 in the original (unreduced) G, but this is easily accomplished in $O(|V| + |E|)$ time.

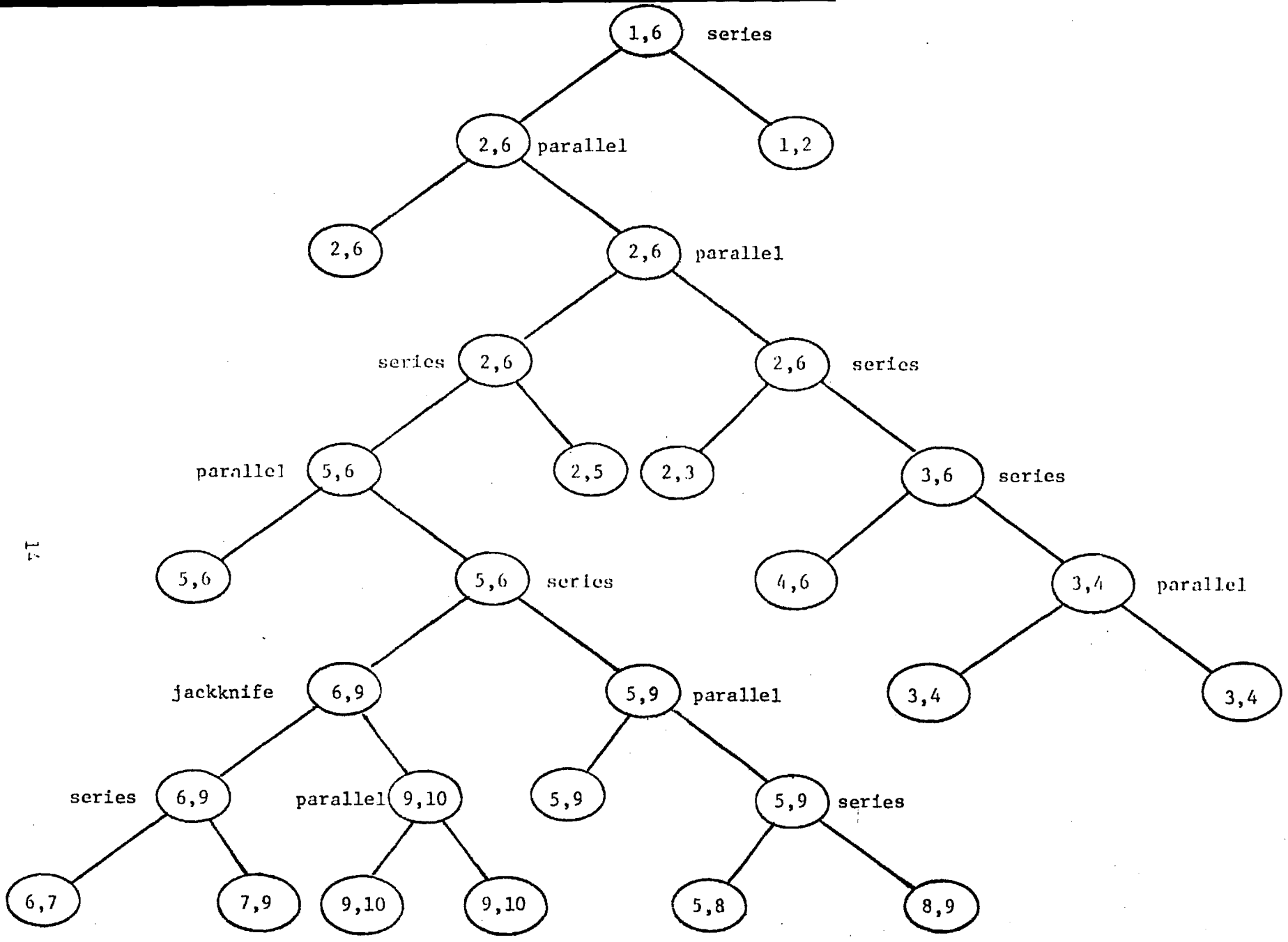


Figure 5: Binary Tree of Reduction Produced for Example of Figure 2

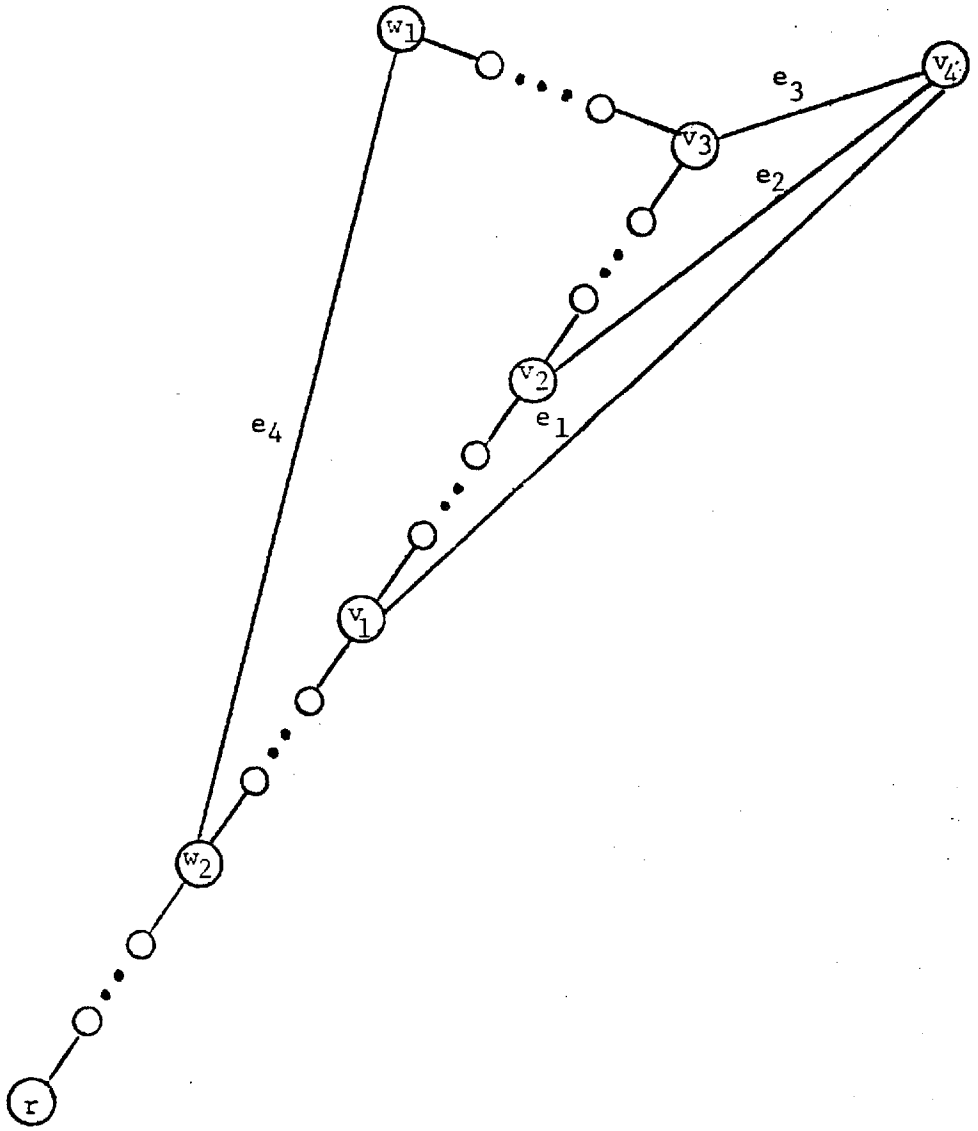


Figure 6. Homeomorphic Subgraph from K_4 Identified at Stage IV

and all edges as unprocessed by

$$b[e] \leftarrow -1 \quad \text{for all } e \in E$$

Then initialize the search at (any degree-1) root vertex r by $\ell \leftarrow 0$, $t[r] \leftarrow 0$, $\ell[r] \leftarrow 0$, $i[r] \leftarrow 1$, $h \leftarrow r$.

Step 1: Process an Edge at h. If $i[h] > d[h]$ go to Step 2 and backtrack. Otherwise, define $e \triangleq s[h, i[h]]$ and $k \triangleq$ the end of e other than h.

1a: If e is an unprocessed back edge ($b[e] < 0$ and $\ell[k] \geq 0$), set $b[e] \leftarrow \ell[k]$ and swap as necessary so that $u[e] = h$, $v[e] = k$. Also, if $h \neq r$, update $b[t[h]] \leftarrow \min\{b[t[h]], b[e]\}$.

1b: If e leads to an unvisited vertex k ($d[k] < 0$), initialize vertex k by $\ell \leftarrow \ell+1$, $\ell[k] \leftarrow \ell$, $t[k] \leftarrow \ell$, $b[e] \leftarrow \ell$, $i[k] \leftarrow 0$. Then swap as necessary so that $u[e] = h$, $v[e] = k$, and advance to k by $h \leftarrow k$.

Set $i[h] \leftarrow i[h]+1$ and repeat Step 1.

Step 2: Backtrack. Define $k \triangleq u[t[h]]$. If $k = r$, Stage I is complete. If not, update $b[t[k]] \leftarrow \min\{b[t[k]], b[t[h]]\}$ and decline to k by $\ell \leftarrow \ell-1$, $h \leftarrow k$. There, advance $i[h] \leftarrow i[h]+1$ and return to Step 1.

Stage II: Radix Sort of Stars

Step 0: Initialization. For $d=1,2,\ldots, |V|$, initialize list $L[d] \leftarrow \phi$. Also set $n[v] \leftarrow 0$ for all $v \in V$.

Step 1: List Loading. For e=1,2,...,$|E|$, add edge e to list $L[b[e]]$.

Step 2: List Unloading. For d = $|V|$, $|V|$-1, ..., 1 unload list $L[d]$ into stars by for each $e \in L[d]$ setting $n[u[e]] \leftarrow n[u[e]]+1$, $s[u[e], n[u[e]]] \leftarrow e$.

Stage III: Reduction

Step 0: Initialization. Initialize the next edge pointer, $\bar{g} \leftarrow |E|$, and edge tree variables $p[e] \leftarrow 0$, $c[e,1] \leftarrow 0$, $c[e,2] \leftarrow 0$ for all $e \in |E|$. Then start a new search at root vertex r via $i[r] \leftarrow 1$, $h \leftarrow r$.

Step 1: Edge Processing. If $i[h] > n[h]$, go to Step 2 and backtrack. Otherwise, define $e \triangleq s[h,i[h]]$, $k \triangleq v[e]$.

1a: If e is a back edge parallel to t[h] ($e \neq t[k]$, $v[e] = u[t[h]]$), advance $i[h] \leftarrow i[h]+1$ and go to Step 4 to parallel reduce with $\bar{e} \leftarrow e$, $\bar{f} \leftarrow t[h]$, $\bar{h} \leftarrow u[t[h]]$.

1b: If e is a back edge not parallel to t[h] and there is no passed over edge at h ($e \neq t[k]$, $o[h] = 0$), advance $i[h] \leftarrow i[h]+1$ and make e the passed over back edge via $o[h] \leftarrow i[h]$.

1c: If e is a back edge parallel to the passed over edge $f \triangleq s[h,o[h]]$ ($e \neq t[k]$, $v[e] = v[f]$), advance $o[h] \leftarrow i[h]$, $i[h] \leftarrow i[h]+1$, and go to Step 4 to parallel reduce with $\bar{e} \leftarrow e$, $\bar{f} \leftarrow f$, $\bar{h} \leftarrow h$.

1d: If e is a back edge parallel to neither t[h] nor the passed over edge at h, G contains a subgraph homeomorphic from K_4. Proceed to Stage IV.

1e: If e is a tree edge and k is a degree-1 vertex (e = t[k], d[k] = 1) advance i[h] ← i[h]+1, and go to Step 5 to jackknife reduce with $\bar{e} \leftarrow e$, $\bar{f} \leftarrow t[h]$, $\bar{h} \leftarrow u[t[h]]$.

1f: If e is a tree edge, k is a degree-2 vertex and $f \triangleq s[k,n[k]]$ is not parallel to e (e = t[k], d[k] = 2, $v[f] \neq u[e]$) go to Step 3 and series reduce with $\bar{e} \leftarrow e$, $\bar{f} \leftarrow f$, $\bar{h} \leftarrow h$.

1g: If e is a tree edge (e = t[k]) and either d[k] > 2 or $f \triangleq s[k,n[k]]$ is parallel to e (v[f] = u[e]), advance to vertex k by h ← k, o[h] ← 0, i[h] ← 1.

Repeat Step 1.

Step 2: Backtracking. Define $e \triangleq t[h]$, $k \triangleq u[e]$.

2a If k = r, stop; G is series-parallel because it has been reduced to the single edge e.

2b: If $k \neq r$, decline to vertex k by h ← k, and return to Step 1.

Step 3: Series Reduction. Advance $\bar{g} \leftarrow \bar{g}+1$ and series reduce edge $\bar{e}$ and $\bar{f}$ into $\bar{g}$ by

$u[\bar{g}] \leftarrow u[\bar{e}]$

$v[\bar{g}] \leftarrow v[\bar{f}]$

$s[\bar{h},i[\bar{h}]] \leftarrow \bar{g}$

$d[v[\bar{e}]] \leftarrow d[v[\bar{e}]]-2$

$p[\bar{e}] \leftarrow \bar{g}$

$p[\bar{f}] \leftarrow \bar{g}$

$c[\bar{g},1] \leftarrow \bar{e}$

$c[\bar{g},2] \leftarrow \bar{f}$

$t[v[\bar{f}]] \leftarrow \bar{g}$ if $\bar{f} = t[v[\bar{f}]]$

Then return to Step 1.

Step 4: Parallel Reduction. Advance $\bar{g} \leftarrow \bar{g}+1$ and parallel reduce $\bar{e}$ and $\bar{f}$ into $\bar{g}$ by

$u[\bar{g}] \leftarrow u[\bar{f}]$

$v[\bar{g}] \leftarrow v[\bar{f}]$

$s[\bar{h},i[\bar{h}]] \leftarrow \bar{g}$

$d[u[\bar{g}]] \leftarrow d[u[\bar{g}]]-1$

$d[v[\bar{g}]] \leftarrow d[v[\bar{g}]]-1$

$p[\bar{e}] \leftarrow \bar{g}$

$p[\bar{f}] \leftarrow \bar{g}$

$c[\bar{g},1] \leftarrow \bar{e}$

$c[\bar{g},2] \leftarrow \bar{f}$

$t[v[\bar{f}]] \leftarrow \bar{g}$ if $\bar{f} = t[v[\bar{f}]]$

Then if a new degree-2 vertex has been created other than at search vertex h ($d[\bar{h}] = 2$, $\bar{g} = t[v[\bar{g}]], d[v[\bar{g}]] > 1$), go to Step 3 and series reduce the new degree-2 vertex via $\bar{e} \leftarrow t[\bar{h}], \bar{f} \leftarrow g, \bar{h} \leftarrow u[\bar{e}]$. If no such vertex was created, return to Step 1.

Step 5: Jackknife Reduction. Advance $\bar{g} \leftarrow \bar{g}+1$ and jackknife reduce $\bar{e}$ and $\bar{f}$ into $\bar{g}$ by

$u[\bar{g}] \leftarrow u[\bar{f}]$

$v[\bar{g}] \leftarrow v[\bar{f}]$

$s[\bar{h},i[\bar{h}]] \leftarrow \bar{g}$

$d[u[\bar{e}]] \leftarrow d[u[\bar{e}]]-1$

$d[v[\bar{e}]] \leftarrow d[v[\bar{e}]]-1$

$p[\bar{e}] \leftarrow \bar{g}$

$p[\bar{f}] \leftarrow \bar{g}$

$c[\bar{g},1] \leftarrow \bar{e}$

$c[\bar{g},2] \leftarrow \bar{f}$

$t[v[\bar{f}]] \leftarrow \bar{g}$ if $\bar{f} = t[v[\bar{f}]]$

Then return to Step 1.

Stage IV: Homeomorphic Subgraph Identification

Step 0: Initialization. Save the search vertex h, the current search edge e, and associated entities on which Stage III terminated as elements of the homeomorphic subgraph depicted in Figure 6. Specifically,

$e_1 \leftarrow e$

$e_2 \leftarrow s[h,o[h]]$

$e_3 \leftarrow t[h]$

$v_1 \leftarrow v[e_1]$

$v_2 \leftarrow v[e_2]$

$v_3 \leftarrow u[e_3]$

$v_4 \leftarrow h$

Then restart the search at the tree predecessor of h by $h \leftarrow v_3$, $i[h] \leftarrow i[h]+1$.

Step 1: Edge Processing. If i[h] > n[h] go to Step 2 and backtrack. Otherwise define $e \triangleq s[h,i[h]], k \triangleq v[e]$.

1a: If e is a back edge touching below v_1 in the tree ($e \neq t[k], \ell[k] \leq \ell[v_1]$, go to Step 3 to complete the forbidden subgraph.

1b: If e is a tree edge ($e = t[k]$

$h \leftarrow k, i[h] \leftarrow 0$.

Increment $i[h] \leftarrow i[h]+1$ and repeat Step 1.

Step 2: Backtracking. Decline to vertex u[t[h]] by setting $h \leftarrow u[t[h]]$. Then return to Step 1.

Step 3: Path Identification. Save $w_1 \leftarrow u[e]$, $w_2 \leftarrow v[e]$, $e_4 \leftarrow e$. Then trace backwards through the tree from w_1 to w_2 by following labels t[k] until a path $w_1,\ldots,v_3,\ldots,v_2,\ldots,v_1,\ldots,w_2$ has been identified. When w_2 is reached stop; this path completes the required homeomorphic subgraph of Figure 6.

■

The principal issue of correctness that must be established for our algorithm is that it stops only with G fully reduced or with a subgraph homeomorphic from K_4.

Lemma 3: Algorithm Stopping. Let G be a connected, undirected, loopless graph with degree-1 vertex r. Then application of the above algorithm to G leads to either termination of Stage III with the correct conclusion that G is series-parallel or termination at Stage IV with a subgraph H'

of the current reduced version G' of G that is homeomorphic from K_4.

Proof.: Consider Stage III of the algorithm. It is easily checked that all reductions undertaken from the different cases of Step 1 are valid series, parallel or jackknife reductions. Furthermore, no vertex is departed via Step 2 until it is either degree-1 or degree-2, i.e. certain to be immediately eliminated by jackknife (respectively series) reduction. Thus if backtracking proceeds until we are ready to return to the root, only r, its incident edge, and the adjacent vertex can remain. (The latter vertex is degree-1 because r is degree-1.) Clearly the final reduced graph G' is a single edge and G is series-parallel.

If, on the other hand, Stage III searching is aborted at Step 1d, entities constructed at Step 0 of Stage IV must be as illustrated in Figure 6. Current search vertex $h = v_4$ is joined to its immediate tree predecessor v_3 by tree edge e_3, and to vertices v_2 and v_1 by the passed over and current back edges e_2 and $e = e_1$. Necessarily v_1, v_2 and v_3 are distinct because otherwise $e = e_1$ would have been parallel reduced. Moreover, we have $\ell[v_1] \leq \ell[v_2]$ because the passed over back edge e_2 was encountered before e_1 in the processing of the Stage-II-ordered star of $h = v_4$.

Most important for the entities recorded at Stage IV, Step 0 is that v_3, the tree predecessor of $h = v_4$, is at least degree-3 and is joined via some descendent vertex w_1 and a back edge e_4 to a vertex w_2 belonging to the tree path from v_1 to r. These claims must hold because had v_3 been degree-2 when first encountered in Stage III it would have been series-reduced by Step 1f, and if it were later made degree-2 by parallel reduction, Step 4 would have passed to Step 3 and series-reduced

it. Moreover, the star ordering introduced at v_3 by Stage II assures that, since the search passed first to e_3 instead of the next listed edge at v_3, the latter edge has a descendent-back-edge path reaching at least as close to the root as the one (e_3,e_1) encountered through e_3. Since the subgraph H' of Figure 6 is clearly homeomorphic from K_4, this completes the proof.

■

Theorem 4: Correctness of the Algorithm. Let G be as in Lemma 3 with vertex set V and edge set E. Then application of the above algorithm to G either exhibits a sequence of series, parallel and/or jackknife reductions converting G to a single edge or produces a subgraph H' of a series, parallel and/or jackknife reduced form G' of G such that H' is homeomorphic from K_4. Moreover, all computation is accomplished in time linear in $|V|$ and $|E|$ ($O(|V|+|E|)$).

Proof: Lemma 3 established all needed properties for correct convergence of the algorithm that do not follow automatically for the nature of depth-first search. To complete the theorem we need only to show computation is in the worst case $O(|V|+|E|)$. We analyze computation by stages:

Stage I. The depth-first search of Stage I begins with $O(|V|+|E|)$ initialization of Step 0. It encounters each edge of E twice at Step 1, once at each vertex of the edge. The $O(|V|)$ tree edges are also backtracked through once each by Step 2. Since all processing is clearly in constant time, computation for Stage I totals $O(|V|+|E|)$.

Stage II. Stage II begins with $O(|V|)$ initialization and then unloads and reloads each edge of E. Total time is $O(|V|+|E|)$.

Stage III. Initialization of Stage III is $O(|E|)$. As the graph is examined each edge is either a tree edge, a back edge saved as the passed over edge, a back edge leading to an immediate parallel reduction, or a back edge causing processing to pass to Stage IV. The $O(|V|)$ tree and passed over back edges are either series or jackknife reduced at once, or so reduced upon backtracking at Step 2. Moreover, each reduction replaces 2 edges by 1 so there are at most $O(|E|)$ reductions. We can conclude Stage III requires at most $O(|V|) + O(|E|)$ edge searching plus $O(|E|)$ reductions. Since all component calculations clearly require constant time, total effort is $O(|V|+|E|)$.

Stage IV. Stage IV requires at most $O(|E|)$ edge examinations and $O(|V|)$ backtracks before it reaches Step 3. There an $O(|V|)$ backtrack through the tree completes processing and total time is $O(|V|+|E|)$.

5. Terminal Subgraphs

Clearly any edge created by series, parallel or jackknife reduction represents a subgraph of the original G. Denote the graph represented by e in a reduced graph $\bar{G}$ as G[e]. If e belonged to the original G, $G[e] \triangleq e$.

We call G[e] a terminal subgraph of G because it is characterized by terminal vertices u[e] and v[e]. These vertices are the only ones of G[e] that persist in reduced graph $\bar{G}$. That is, G[e] is joined to the

rest of G only through terminal vertices u[e] and/or v[e].

Reversing this perspective, we can develop an equivalent characterization of series-parallel graphs in terms of terminal subgraph separations . Cut vertices and cut pairs were defined in Section 3. A terminal graph G[u,v] is any graph with distinct vertices u and v identified as terminals. We consider the separation of loopless terminal multigraphs G[u,v] containing no isolated vertices into two loopless terminal multisubgraphs $G_1[u_1,v_1]$ and $G_2[u_2,v_2]$ containing no isolated vertices and satisfying $G[u,v] = E(G_1[u_1,v_1]) \cup E(G_2[u_2,v_2])$.

- series separation: separate G[u,v] at a cut vertex $c \neq u,v$ into terminal subgraphs $G_1[u,c]$ and $G_2[c,v]$, such that $V(G_1[u,c]) \cap V(G_2[c,v]) = \{c\}$.
- parallel separation: separate G[u,v] at (terminal) cut pair into terminal subgraphs $G_1[u,v]$ and $G_2[u,v]$, such that $V(G_1[u,v]) \cap V(G_2[u,v]) = \{u,v\}$.
- jackknife separation: separate G[u,v] at (terminal) cut vertex u (respectively (terminal) cut vertex v) into terminal subgraphs $G_1[u,v]$ and $G_2[u,t]$ (respectively $G_2[t,v]$) satisfying $V(G_1[u,v]) \cap V(G_2[u,t]) = \{u\}$ (respectively $V(G_1[u,v]) \cap V(G_2[t,v]) = \{v\}$) and t is any vertex of G_2 except u or v.

Theorem 5: Separation Characterization. A loopless, undirected multigraph G with no isolated vertices is series-parallel if and only if every connected component of G can be separated into a collection of disjoint edges by designating an appropriate pair of distinct vertices of the component as terminals and applying some sequence of series, parallel and jackknife separations.

Proof: First assume G is series-parallel and consider a connected component $\bar{G}$. By applying the algorithm of Section 4 we can construct a binary tree of reductions with corresponding terminal subgraphs such that G[g] is the parent of G[e] and G[f] in the tree if e and f were series or parallel or jackknife-reduced to g. Ends of this binary tree are single edges, and the last-constructed, root vertex of this binary tree has $G[g] = \bar{G}$.

Viewing this binary tree from its root, we want to show it corresponds exactly to a sequence of series, parallel and jackknife separations leading to a disjoint collection of edges. We begin by choosing as terminals of the root graph the two vertices of the edge to which it was reduced. Now proceed inductively through the binary tree. If a G[g] was formed by series reduction of G[e] and G[f], each of G[e] and G[f] has exactly one terminal in common with G[g], and $V(G[e]) \cap V(G[f])$ is their common terminal (which is not a terminal of G[g]). These are exactly the requirements for a series separation.

If G[g] was formed by parallel reduction of G[e] and G[f], all three have the same terminals and $V(G[e]) \cdot V(G[f]) = \{u[e], v[e]\}$. Reversal of the reduction is a parallel separation.

Finally, suppose G[g] was created by jackknife reduction of G[e] and G[f]. G[e] and G[f] intersect only at their common terminal, G[f] has the same terminals as G[g], and G[e] has as terminals the one it shares with G[f] and some other vertex. Clearly G[g] jackknife separates into G[e] and G[f].

For the converse we apply Theorem 1. If G is not series-parallel we know from the earlier result that G has a connected component $\bar{G}$ with a

subgraph $\bar{H}$ homeomorphic for K_4. We will show this implies no sequence of series, parallel and jackknife separations can divide $\bar{G}$ into a collection of disjoint edges.

Consider a sequence of series, parallel and jackknife separations of $\bar{G}$, and let G_0 be the last one containing every edge of $\bar{H}$. That is $E(\bar{H}) \subset E(G_0[u,v])$, but separation of $G_0[u,v]$ into $G_1[u_1,v_1]$ and $G_2[u_2,v_2]$ will leave $E(\bar{H}) \not\subset E(G_1)$, $E(\bar{H}) \not\subset E(G_2)$. Now $\bar{H}$ is homeomorphic from K_4, so it contains no cut vertices. Also, cut pairs are possible only for pairs of vertices both belonging to a path of $\bar{H}$ corresponding to a single edge of K_4. It follows that the separation was of the parallel type, that both terminals of G_0, G_1 and G_2 belong to such a path of $\bar{H}$, and that one of G_1 and G_2 contains all of $\bar{H}$ except a path through degree-2 vertices. That is, one of G_1 and G_2, say G_1, contains a subgraph of the form $\bar{H}$ depicted in Figure 7.

Further parallel separation of G_1 will leave terminals unaltered and $\bar{H}'$ entirely in one or the other created subgraph. Jackknife separations are possible if u or v is a cut vertex, but again, all of $\bar{H}'$ must belong to one of the successors and terminals will be undisturbed. Series separations do move terminals, but only to cut vertices dividing the graph so that one original terminal belongs to each successor.

We can conclude that only with series separations can a subsequent sequence of series, parallel and/or jackknife reductions not leave $\bar{H}'$ entirely contained in one of the two resultant subgraphs. But all possible cut vertices of a graph like $\bar{H}'$ belong to either the path (a,...,u) or the path (v,...,c). It follows that any sequence of such separations will eventually lead to a descendent containing the subgraph $\bar{H}''$ of Figure 7.

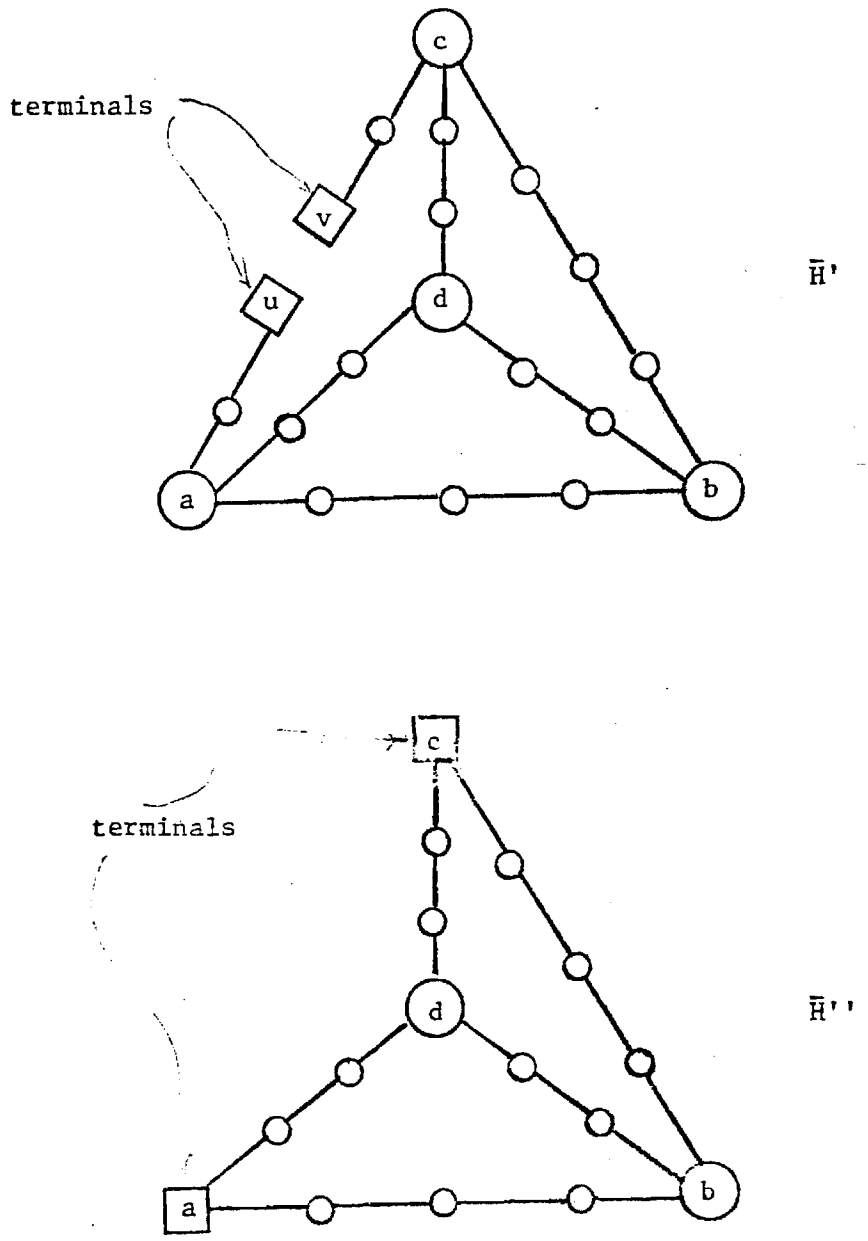


Figure 7: Failure of Separation on Subgraphs Homeomorphic from K_4

Further separation of $\bar{H}''$ is impossible. Thus, we can conclude that if $\bar{G}$ originally contained an $\bar{H}$ homeomorphic from K_4, $\bar{G}$ cannot be separated by any sequence of series, parallel and/or jackknife separations into a collection of disjoint edges. This completes the proof.

6. Comparison to Other Definitions

The fundamental Duffin (1965) paper on series-parallel graphs sought to clarify relations between alternative definitions in the earlier work of Riordan and Shannon (1942). Duffin defined series-parallel graphs as those for which resistance between any adjacent pair of terminals could be computed by Ohm's laws:

- Resistance is additive for resistors in series
- Reciprocal resistance is additive for resistors in parallel

Duffin also defined confluent graphs as graphs having no cycles C_1 and C_2 that cannot be oriented in such a way that all common edges have like direction.

Any distinct 4-vertex cycles of K_4 fail the confluence property. More generally Duffin proved (his Theorem 1) that a graph is confluent if and only if it contains no subgraph homeomorphic from K_4. Furthermore, in his Theorem 3 he established that a graph is series-parallel if and only if it is confluent.

We can thus conclude via our Theorem 1 that both our definition and Duffin's definition of series-parallel graphs are equivalent. However, Duffin defined only two reductions--equivalent to our series and parallel. Consequently, he only gave a reduction characterization of the case

where G is biconnected. The simple graph of Figure 3 illustrates that cases lacking a subgraph homeomorphic from K_4 but still not completely reducible by series and parallel operations, are possible when G is not biconnected.

The recent paper by Valdes, Tarjan and Lawler (1982) is primarily concerned with a vertex form of series-parallel graphs. However, these authors do define and employ edge series-parallel digraphs. Lemma 2 of their work references Duffin in asserting digraphs are edge-series-parallel if and only if they can be reduced to single edges by series and parallel reductions. Thus their definition is limited to graphs reducible by series and parallel operations alone, and excludes, for example, the graph of Figure 3.

Another recent and important paper on series-parallel graphs is that of Takamizawa, Nishizeki and Saito (1982). These authors define series-parallel graphs as those reducible by series and parallel reductions to a two-edge cycle. It is easy to see that this limits their series-parallel graphs to the biconnected ones since series and parallel reductions preserve biconnectivity.

More importantly, however, Takamizawa et al treat a more general form called two-terminal series-parallel. Like our Section 5, two-terminal series-parallel graphs are defined in terms of separations of a given graph G. The process begins with up to two vertices of G designated as terminals. New graphs produced by separating G also have two terminals. However, one or both may be virtual (i.e. artificial). If we adopt the equivalent notion that single terminal and no terminal graphs are allowed, two terminal series-parallel graphs are those which can be decomposed into a collection of disjoint edges by any sequence of application of the series (type I) and parallel separations tabulated in our Table 1.

Table 1: Classification of Separations in Two-Terminal Series-Parallel Graphs[1/]

Separation Form	Separation Vertex(s)	Key G[2/] Vertices	Key G_1[2/] Vertices	Key G_2[2/] Vertices
Series (Type I)	b	a,b,c	a,b	b,c
	b	a,b	a,b	b
	b	b	b	b
Parallel	a,b	a,b	a,b	a,b
	a	a,b	a,b	a,b
	none	no terminals	no terminals	no terminals

1/ Adapted from Takamizawa et al (1982)

2/ Underlined vertices are terminals.

Series (type I) separations divide G at a cut-vertex, c, into two parts G_1 and G_2. The cut-vertex c is not a terminal of G but becomes a terminal of both G_1 and G_2. Terminals of G are also terminals of whichever of G_1 and G_2 they belong to. In particular, if G had two terminals one must be part of G_1 and the other part of G_2.

Parallel separations have three purposes. If G has no terminals, parallel separation merely divides G into one collection G_1 and another G_2 of disconnected components of G. If G has one terminal and it is a cut-vertex, parallel separation creates new one-terminal subgraphs separated at that cut-vertex. If G has two terminals, parallel separation divides G at the terminals when they form a cut-pair.

Clearly, series (type I) and parallel separations with two terminals perform substantially the same functions as our series and parallel separations of Section 5. Parallel separations with no terminals merely isolate connected components.

Parallel separations with one terminal are more interesting. They effect part of what we do by jackknife separation -- divide graphs at a cut-vertex that is also a terminal. However, two-terminal series-parallel graphs still do not appear to cover the full range of our series-parallel graphs. It is not hard to verify that separation of the graph in Figure 3 by methods of Table 1 eventually halts with a subgraph consisting of a 3 vertex path having 2 adjacent terminal vertices. Thus the graph of Figure 3 is not two-terminal series-parallel even though it conforms to our definition of series-parallel. On the other hand, a proof like that of the converse of Theorem 5 can show that if G is not (our) series-parallel, it is not two-terminal, series-parallel. The subgraph $\bar{H}''$ of

Figure 7 admits no Table 1 separation. But just as in the proof of Theorem 5, it must eventually be encountered. Whether we start with 0, 1 or 2 terminals, $\bar{H}$ can be first disturbed only by the two-terminal, cut-pair form of parallel separation at which our proof begins.

Finally, we note that our own work in Rardin, Parker, and Wang (1982) and Rardin, Parker, and Richey (1982) use still a different definition of series-parallel graphs. We described such graphs as those reducible by series and parallel reduction to a tree. Clearly, the example of Figure 3 fails this test even though it satisfies the definition of Section 2.

7. Conclusion

Our aim in this paper has been to clarify and synthesize knowledge about series-parallel graphs. We can summarize our conclusions by the following:

Theorem 6: Equivalent Characterizations. Let G be a loopless, undirected multigraph with no isolated vertices. Then the following are equivalent:

(i) G can be reduced to a collection of disjoint edges by some sequence of applications of series, parallel and jackknife reductions (as defined in Section 2).

(ii) G contains no subgraph homeomorphic from K_4.

(iii) G contains no subgraph homeomorphic from any triconnected graph.

(iv) Every connected component of G can be separated into a collection of disjoint edges by designating an appropriate pair of vertices of the component as terminals and applying

some sequence of series, parallel and jackknife separations (as defined in Section 5).

(v) G contains no cycles C_1 and C_2 that cannot be oriented in such a way that all common edges have like direction.

Furthermore, whether G satisfies (i) - (iv) can be tested in time linear in the number of its edges and vertices.

REFERENCES

Aho, A. V., J. E. Hopcroft and J. D. Ullman (1974), The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading MA.

Duffin, R.J. (1965), "Topology of Series-Parallel Networks", Journal of Mathematical Analysis and Applications, 10, 303-318.

Hopcroft, J.E. and R. E. Tarjan (1973), "Dividing a Graph into Triconnected Components", SIAM Journal on Computing, 2, 135-158.

Nishizeki, T., K. Takamizawa and N. Saito (1976), "Algorithms for Detecting Series Parallel Graphs and D-Charts", Trans. Inst. Elect. Commun. Eng. Japan, 59, 259-260.

Rardin, R.L., R.G. Parker and W.Y. Lim (1982), "Some Polynomially Solvable Multi-Commodity Fixed Charge Network Flow Problems", Industrial and Systems Engineering Report Series J-82-2, Georgia Institute of Technology, March.

Rardin, R.L., R.G. Parker and M.B. Richey (1982), "A Polynomial Algorithm for a Class of Steiner Tree Problems on Graphs", Industrial and Systems Engineering Report Series J-82-5, Georgia Institute of Technology, August.

Riordan, J. and C. E. Shannon (1942), "The Number of Two-Terminal Series-Parallel Networks", J. Math. Phys., 21, 83-93.

Takamizawa, K., T. Nishizeki and N. Saito (1982), "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs", Journal of the ACM, 29, 623-641.

Valdes, J. (1978), "Parsing Flowcharts and Series-Parallel Graphs," Technical Report STAN-CS-78-682, Stanford University.

Valdes, J., R.E. Tarjan and E.L. Lawler (1982), "The Recognition of Series Parallel Digraphs", SIAM J. Computing, 11, 298-313.

Industrial and Systems Engineering
Report Series J-82-5
August, 1982

A POLYNOMIAL ALGORITHM FOR A CLASS OF STEINER TREE PROBLEMS ON GRAPHS

by

Ronald L. Rardin

R. Gary Parker

Michael B. Richey

J-82-5

This material is based in part upon work partially supported by the National Science Foundation under Grant Number ECS-8018954.

ABSTRACT

In this paper, we address the following problem. Given an undirected graph $G(V,E)$ with arbitrary edge weights, determine a minimal weight subset of edges which forms a tree and which includes a specific subset of vertices in V. It is well known that finding such a subgraph, referred to as a Steiner tree, is formally difficult for arbitrary $G(V,E)$. On the other hand, if we confine our attention to a class of graphs commonly referred to as series-parallel, the problem can be solved. We demonstrate this in the present paper.

1. INTRODUCTION

Consider an undirected graph $G(V,E)$ with vertex and edge sets V and E respectively. We shall assume throughout that $G(V,E)$ is loopless but may possess multiple edges. The Steiner tree problem on graphs seeks a subset $\hat{E} \subseteq E$ forming a tree which includes all vertices in a specified subset $S \subseteq V$ and which has minimum total edge weight. Note that we specifically define the problem on graphs in order to differentiate it from the classic version defined on the Euclidean plane. In the latter case, the problem is a well-known one in geometry which asks for a set of lines (also in the plane) which connect a set of points and does so with minimum total length. Hereafter, we will simply refer to our problem by STP.

The difficulty of the STP has been established, its *NP*-Completeness (for the decision analog) following via a transformation from EXACT COVER BY 3-SETS (Karp (1972)). Even various special cases remain intractable. Among these are problems with equal edge weights (Garey and Johnson (1979)) as well as problems defined on planar graphs (Garey and Johnson (1977)).

On the other hand, there are some easily resolvable cases. Clearly, when $S = V$, the STP trivially reduces to the minimum weight spanning tree problem for which highly efficient algorithms are known and when $|S| = 2$, the problem becomes one of finding the shortest path in a graph which connects the vertices in S. This also is well-solved so long as edge weights are non-negative. Of course, general (nonpolynomial) algorithms have also been developed. We direct the interested reader to Lawler (1976) for coverage of these, in particular the work of Dreyfus and Wagner (1972). An interesting survey of the STP can also be found in Hakimi (1971) and recently, a branch and bound procedure was presented by Shore, et. al., (1982).

In this paper, we give a procedure for solving the STP on a restricted class of graphs referred to as series-parallel. Our algorithm is based, in

part, on notions developed in Takamizawa, et. al. (1982) as well as on implications arising from earlier results in Rardin, et. al. (1982). Following, we characterize the class of series-parallel graphs of interest after which we motivate the development of the algorithm. The procedure is formally stated and demonstrated with a small problem. We next establish the algorithm's polynomiality and finally, we conclude with some general observations.

2. SERIES-PARALLEL GRAPHS

The notion of series-parallelism in graphs is not of recent vintage. It also is a concept which has found its way into a host of problem settings dealing with graphs and networks. Included are classic problems in electrical engineering, routing and transportation problems, problems in network design and various precedence constrained scheduling problems.

Accompanying this range of settings has been a concomittant growth in the literature of the various fields. This, in turn, has created a somewhat confusing situation insofar as a unified view of series-parallel structures is concerned. An early attempt at resolving this appeared in the work of Duffin (1965), and later, in Lawler (1978).

For our purposes, the following definition of series-parallel graphs will suffice. In particular, we will say that a loopless, undirected graph, $G(V,E)$ is series-parallel if it can be reduced to a forest by the sequential application of the following elementary operations.

(i) series reduction: Replace any degree-2 vertex k, and the incident edges (or pseudo-edges) e and f connecting k to vertices i and $j \neq i$ respectively, by a new pseudo-edge g incident to i and j.

(ii) parallel reduction: Replace two edges (either or both of which may be pseudo-edges) e and f, both incident to vertices i and j, by a new pseudo-edge g incident to i and j.

An alternative, and more familiar definition suggests that $G(V,E)$ is series-parallel if and only if it possesses no subgraph homeomorphic to K_4 (the complete graph on four vertices). This definition assures that all conforming graphs are planar but not the converse since K_4 itself is planar. Regardless, it is easy to show that these two views of the series-parallel property are not equivalent, e.g., any 1-tree satisfying the latter definition and which has no degree-2 vertices is not reducible by operations (i) and (ii) above.

With no additional specifications on $G(V,E)$, our class of series-parallel graphs can accurately be viewed as a restricted class of those conforming to the definition employing the forbidden K_4 subgraphs (or homeomorphs of K_4). However, if we take our graphs to be 2-connected then the definitions are equivalent. Recall that a graph is 2-connected if every pair of vertices lies on a cycle. Further, under 2-connectivity any valid sequence of operations (i) and (ii) will reduce such a graph (if and only if it is series-parallel) to a single edge or pseudo-edge ([2]).

Note that we have introduced the term pseudo-edge in order to signify artificial edges which result from series and parallel reductions. These artificial edges, of course, represent subgraphs of $G(V,E)$. Let us denote by $G(e)$ the subgraph associated with pseudo-edge e and by $V(e)$ and $E(e)$, the vertex and edge sets of $G(e)$, respectively. The vertices to which e is incident will be termed the terminals of $G(e)$.

When $e \in E$ (i.e., e is a "true" edge in G) and e is incident to vertices say i and j, we shall set $V(e) \leftarrow \{i, j\}$ and $E(e) \leftarrow \{e\}$ where $e \triangleq (i, j)$. When e and f are series or parallel reduced to g, the associated G(g) is obtained via $V(g) \leftarrow V(e) \cup V(f)$ and $E(g) \leftarrow E(e) \cup E(f)$. The key observations leading to the development of an algorithm for the STP on series-parallel graphs can now be stated.

Lemma 1: Common Elements of Series or Parallel-Reduced Subgraphs. Suppose edges or pseudo-edges e and f are series or parallel-reduced to pseudo-edge g. Then

1a: $E(e) \cap E(f) = \phi$

1b: $V(e) \cap V(f) = \{k\}$ where k is the common terminal of G(e) and G(f), if the reduction is series.

1c: $V(e) \cap V(f) = \{i, j\}$ where i and j are the two terminals of G(e) and G(f), if the reduction is parallel.

Proof: All edges and all non-terminal vertices of G(e) and G(f) have been absorbed in one of e or f before the two are combined as g. Hence, no such edge or non-terminal vertex could belong to both G(e) and G(f). In addition, G(e) and G(f) have exactly one (for series) or two (for parallel) vertices in common which follows by definition of series and parallel reduction. ∎

It is obvious that the series and parallel reduction operations neither create not join disconnected components of G. Thus reduction will lead to a forest which is a tree exactly when G is connected. In dealing with the STP, we need consider only this case since $G(V,E)$ admits a Steiner tree on vertices $S \subseteq V$ if and only if vertices in S belong to a single component.

Earlier we stated that under the assumption that the original graph is 2-connected, the reduction process always leads to a single (pseudo) edge. In order to simplify the ensuing presentation of cases, we will assume such a reduction to an edge always occurs. In cases where reduction yields a tree of 2 or more edges (the original graph may even be a tree) we can induce 2-connectivity by adding an artificial, non-Steiner vertex, $\hat{v}$, to the current, reduced graph and connecting it to every vertex of the tree by artificial edges having weight $+\infty$. Clearly, such an edge and hence, $\hat{v}$, will not be part of an optimal Steiner tree. The next lemma assures that this construction results in a 2-connected graph and more importantly, that it preserves the series-parallel property.

Lemma 2: Artificial 2-Connection of Tree Cases. Let T be a tree on vertex set $V(T)$. Then the graph H formed by connecting a new vertex $\hat{v}$ to each $i \in V(T)$ is 2-connected, series-parallel.

Proof: That H is 2-connected is clear since every pair $i, j \in V(T)$ lies on the cycle formed by edges $(\hat{v}, i)$, $(\hat{v}, j)$ and the unique path in T connecting i and j. To see that H is also series-parallel observe that if this were not so, then H would necessarily contain a subgraph homeomorphic to K_4. One vertex in such a subgraph might be $\hat{v}$ but at least three would belong to $V(T)$. However, this would imply the presence of K_3 (or a homeomorph of K_3) in T and hence, a cycle, which is not possible if T is a tree. Thus H contains no such homeomorph and is indeed series-parallel.

■

We are now in a position to develop an algorithm for the STP under the stated series-parallel assumption.

3. DEVELOPMENT OF AN ALGORITHM

Our aim is to develop an inductive procedure. That is, we wish to express optimal Steiner solutions in a subgraph, say G(g). This subgraph results from a series or parallel reduction as a function of its antecedent subgraphs, say G(e) and G(f). From Lemma 1, we observed that G(e) and G(f) share no edges and have only terminal vertices in common. As a consequence, there are relatively few cases to consider. The next lemma lists the possibilities for a series reduction.

Lemma 3: Antecedents of a Tree Produced by Series Reduction. Suppose degree-2 vertex k, and edges or pseudo-edges e and f connecting k to i and $j \neq i$ respectively, are series-reduced to pseudo-edge g. Then every tree, T, of G(g) satisfies one of the following:

3a: T is a tree of G(e)

3b: T is a tree of G(f)

3c: T is the union of a tree of G(e) and a tree of G(f), both of which include vertex k.

■

To produce a similar classification of cases under parallel reduction, we require an additional concept. B is a terminal biforest of the subgraph G(g) associated with pseudo-edge g if B consists of two disjoint trees, each including exactly one of the terminals of g. Accordingly, we have

Lemma 4: Antecedents of a Tree Produced by Parallel Reduction. Suppose two edges or pseudo-edges e and f, both connecting vertices i and j, are parallel-reduced forming pseudo-edge g. Then every tree, T, of G(g) satisfies one of the following:

4a: T is a tree of G(e).

4b: T is a tree of G(f).

4c: T is the union of trees of G(e) and G(f) both of which contain i and neither of which contains j.

4d: T is the union of trees of G(e) and G(f) both of which contain j and neither of which contains i.

4e: T is the union of a tree of G(e) containing both i and j with a terminal biforest of G(f).

4f: T is the union of a tree of G(f) containing both i and j with a terminal biforest of G(e).

■

Having introduced the terminal biforest cases, we must now consider how they might occur in reductions. The next two lemmas treat the series and parallel cases.

Lemma 5: Antecedents of a Terminal Biforest Produced by Series Reduction. Suppose degree-2 vertex k, and edges or pseudo-edges e and f connecting k to i and j ≠ i respectively, are series reduced to pseudo-edge g. Then every terminal biforest, B, of G(g) satisfies one of the following.

5a: B is the union of a tree of G(e) including vertex i but not k, and a tree of G(f) including vertex j but not k.

5b: B is the union of a tree of G(e) including both vertices i and k and a terminal biforest of G(f).

5c: B is the union of a tree of G(f) including both vertices j and k and a terminal biforest of G(e).

■

Lemma 6: Antecedents of a Terminal Biforest Produced by Parallel Reduction. Suppose two edges or pseudo-edges e and f, both connecting vertex i to vertex j, are parallel reduced to a new pseudo-edge g. Then every terminal biforest, B, of G(g) satisfies the following.

6a: B is the union of terminal biforests of G(e) and G(f).

■

The previous four lemmas demonstrate that trees and terminal biforests of new pseudo-edge subgraphs can be derived from similar results which exist relative to their antecedents. We need only exercise some care in recording which terminals belong to various trees. To specialize the results to Steiner trees and Steiner terminal biforests (terminal biforests for which all Steiner vertices belong to one of the two trees), we need only check whether the antecedent structures can contain all required Steiner vertices. Also, since structures obtained by union in various cases of Lemmas 3-6 represent the union of edge-disjoint entitites, the optimal union will possess total weight equivalent to the sum of the weight of optimal antecedents of the specified types.

With these observations, an algorithm for the STP can be stated in terms of edge (or pseudo-edge) labels. We define these below.

t(e, -) $\triangleq$ the weight of a minimum Steiner tree on G(e) that uses neither terminal.

t(e, k) $\triangleq$ the weight of a minimum Steiner tree on G(e) that uses only terminal k (one label for each terminal).

t(e, +) $\triangleq$ the weight of a minimum Steiner tree on G(e) that uses both terminals.

t(e, |) $\triangleq$ the weight of a minimum Steiner terminal biforest on G(e).

We can now state the algorithm formally. In the following, we denote the Steiner vertices by set S and edge weights, which may be arbitrary, by w(e) for $e \triangleq (i, j) \in E$.

Step 0: Label Edges in E. To each edge e = (i,j) in E assign the labels

$$t(e, -) \leftarrow +\infty$$

$$t(e, i) \leftarrow \begin{cases} +\infty & \text{if } j \in S \\ 0 & \text{otherwise} \end{cases}$$

$$t(e, j) \leftarrow \begin{cases} +\infty & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

$$t(e, +) \leftarrow w(e)$$

$$t(e, |) \leftarrow 0$$

Step 1: Stopping. If the present graph is a single (pseudo) edge, e = (i,j), stop; the weight of an optimal Steiner tree in G is

$$\min \{ t(e, -), t(e, i), t(e, j), t(e, +) \}.$$

Otherwise, if a series reduction is possible, go to Step 2 and if a parallel reduction is possible, go to Step 3.

Step 2: Series Reduction. Select any degree-2 vertex k of the present graph and let e be the edge (or pseudo-edge) connecting k to i and f, the edge (or pseudo-edge) connecting k to j. Replace k, e and f by a new pseudo-edge, g, with labels

$$t(g,-) \leftarrow \begin{cases} \min\{t(e,k) + t(f,k),\ t(e,-)\} & \text{if } V(e) \cap S \neq \phi \text{ and } V(f) \cap S = \phi \\ \min\{t(e,k) + t(f,k),\ t(f,-)\} & \text{if } V(f) \cap S \neq \phi \text{ and } V(e) \cap S = \phi \\ \min\{t(e,k) + t(f,k),\ t(e,-),\ t(f,-)\} & \text{if } \{V(e) \cup V(f)\} \cap S = \phi \\ t(e,k) + t(f,k) & \text{otherwise} \end{cases}$$

$$t(g,i) \leftarrow \begin{cases} t(e,+) + t(f,k) & \text{if } V(f) \cap S \neq \phi \\ \min\{t(e,i),\ t(e,+) + t(f,k)\} & \text{otherwise} \end{cases}$$

$$t(g,j) \leftarrow \begin{cases} t(f,+) + t(e,k) & \text{if } V(e) \cap S \neq \phi \\ \min\{t(f,j),\ t(f,+) + t(e,k)\} & \text{otherwise} \end{cases}$$

$$t(g,+) \leftarrow t(e,+) + t(f,+)$$

$$t(g,|) \leftarrow \min\{t(e,+) + t(f,|),\ t(e,|) + t(f,+),\ t(e,i) + t(f,j)\}$$

Return to Step 1.

Step 3: Parallel Reduction. Select any two edges or pseudo-edges e and f connecting the same pair of vertices i and j in the present graph. Replace e and f by a new pseudo-edge, g, with labels

$$t(g,-) \leftarrow \begin{cases} t(e,-) & \text{if } V(e) \cap S \neq \phi \text{ and } V(f) \cap S = \phi \\ t(f,-) & \text{if } V(f) \cap S \neq \phi \text{ and } V(e) \cap S = \phi \\ \min\{t(e,-),\ t(f,-)\} & \text{if } \{V(e) \cup V(f)\} \cap S = \phi \\ +\infty & \text{otherwise} \end{cases}$$

$$t(g,i) \leftarrow t(e,i) + t(f,i)$$

$$t(g,j) \leftarrow t(e,j) + t(f,j)$$

$t(g, +) \leftarrow \min \{ t(e,|) + t(f,+),\ t(f,|) + t(e,+) \}$

$t(g, |) \leftarrow t(e,|) + t(f,|)$

Return to Step 1.

■

Prior to demonstrating the algorithm, we establish with the next result the validity as well as the efficiency of the procedure. We have

Theorem: Correctness and Efficiency. Let G be a 2-connected and series-parallel graph. The stated algorithm computes the weight of an optimal Steiner tree in G in time growing as a polynomial in the number of edges and vertices in G.

Proof: To see that the algorithm is polynomial, we observe that each reduction of either type reduces the number of edges and pseudo-edges by one. In addition, computation associated with a reduction involves only scans for reducible cases and within each, label updates. Hence, the algorithm is clearly polynomial in $|V|$ and $|E|$.

To show that the procedure always yields the correct solution value (or $+\infty$ if G has no Steiner tree) we can proceed inductively. Observe that straightforward application of the label definitions establishes that initial assignments are correct.

Now, assume that labels do reflect the desired optimal values for edges or pseudo-edges e and f being series-reduced as in Step 2 of the algorithm. Label rules of Step 2 merely enumerate the antecedent combinations contemplated by Lemmas 3 and 5 in order to produce optimal labels for the new pseudo-edge g. Similarly, if a parallel reduction is performed at Step 3, labeling enumerates the cases of Lemmas 4 and 6. Since these are the only cases, correct labels must result.

■

Of course determining the weight of an optimal Steiner tree is not the same as "solving" the STP. To accomplish this, we must produce an appropriate subgraph having the optimal weight. The issue, however, is no different than that in say, shortest path analysis where a particular path is easily reconstructed from its length calculations by a simple backtracking scheme. The following example serves as an illustration.

Suppose $G(V,E)$ is given as in Figure 1a where edge weights, $w(e)$, are given on the figure. Let $S = \{2, 3, 7\}$ and for ease, denote the edge labels by the format shown in Figure 1b. Proceeding in step-by-step fashion we have

Step 0: Initially, the labels on all edges are set and appear as shown in Figure 2.

Step 1: Consider vertex 6 which is of degree two. Letting $e = (5,6)$ and $f = (6,4)$ we perform a series reduction, replacing e and f by $g = (5,4)$. The labels for g are calculated and we have

$$\begin{aligned} t(g, -) \leftarrow & \min \{ t(e,6) + t(f,6),\ t(e,-),\ t(f,-) \\ & = \min \{ 0 + 0,\ +\infty,\ +\infty \} \\ & = 0 \end{aligned}$$

$$\begin{aligned} t(g, i) \leftarrow & \min \{ t(e,5),\ t(e,+) + t(f,6) \} \\ & = \min \{ 0,\ 10 + 0 \} \\ & = 0 \end{aligned}$$

$$t(g, j) \leftarrow \min \{ t(f,4),\ t(f,+) + t(e,6)\}$$
$$= \min \{0,\ 5 + 0\}$$
$$= 0$$

$$t(g, +) \leftarrow t(e,+) + t(f,+) = 15$$

$$t(g, |) \leftarrow \min \{t(e,+) + t(f,|),\ t(e,|) + t(f,+),\ t(e,i) + t(f,j)\}$$
$$= \min \{0, +\infty, 0\} = 0$$

Step 1: Replacing edges (5,6) and (6,4) by pseudo-edge (5,4) clearly does not produce a single edge graph and we continue.

Step 2: Vertex 5 is now a degree-2 vertex and we can perform a series reduction again. Letting e ← (3,5) and f ← (5,4) in the present graph, we create the new pseudo-edge g ← (3,4). The labels on g are $+\infty$, 0, $+\infty$, 19 and 0 for t(g, −), t(g, i), t(g, j), t(g, +) and t(g, |) respectively.

Step 1, 3: A pair of parallel edges now connect vertices 3 and 4. One of these is a real edge and the other is the result of the previous two series reductions. Letting the pseudo-edge be the latter, denote it by e and the true edge by f. Forming the new pseudo-edge, $g \triangleq (3,4)$ by parallel reduction produces the following labels

$$t(g, -) \leftarrow +\infty \quad \text{(note: } V(e) \text{ and } V(f) \text{ share Steiner vertex 3)}$$

$$t(g, i) \leftarrow t(e,3) + t(f,i)$$
$$= 0$$

$$t(g, j) \leftarrow t(e,4) + t(f,4)$$
$$= +\infty$$

$$t(g, +) \leftarrow \min \{ t(e,|) + t(f,+),\ t(f,|) + t(e,+)\}$$
$$= \min \{0 + 3,\ 0 + 19\}$$
$$= 3$$

$$t(g, |) \leftarrow t(e,|) + t(f,|)$$
$$= 0$$

The procedure continues in this manner until stopping occurs with a single edge. This is guaranteed since the original graph is 2-connected. Regardless, we summarize the entire calculation in Figure 3 where at each iteration the new labels are given along with the associated subgraph G(g). The reader will observe that when Step 1 is finally invoked an optimal weight results as $\min \{ t(e, -), t(e, 8), t(e, 9), t(e, +)\} = \min \{3, 1, 5, -2\} = -2$ where the $e \triangleq (8, 9)$. The optimal tree is shown in Figure 4.

■

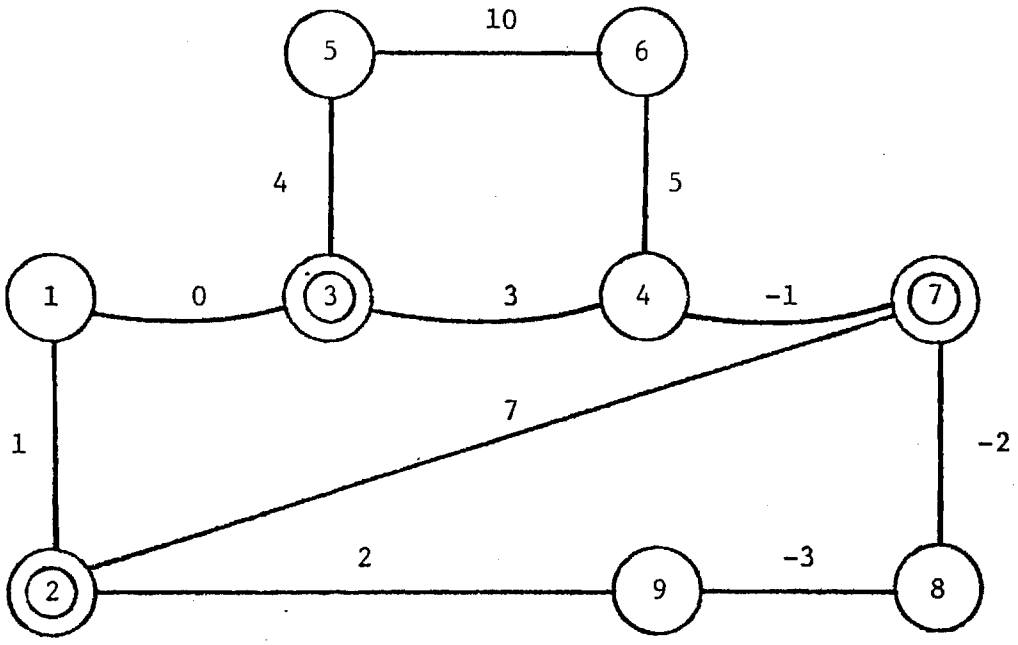
4. SUMMARY AND CONCLUSIONS

In this paper we have presented a polynomial algorithm for treating the Steiner tree problem defined on graphs which possess a series-parallel structure. When our graphs are 2-connected-series-parallel on equivalent characterization is that they contain no subgraph homeomorphic to K_4.

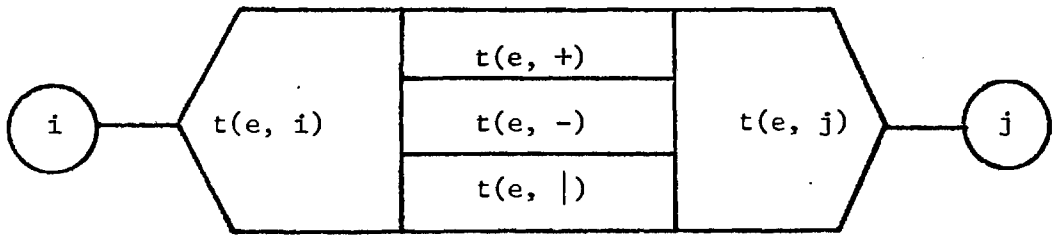
This work on Steiner trees stems directly from a more general context in which it has been shown by the first two authors that a rather rich class of combinatorial problems are efficiently solvable when the series-parallel property is present. In particular, it is known that numerous such problems

can be formulated as multi-commodity fixed charge network flow problems, the linear programming relaxation of which is perfect, i.e., integer. This latter condition results from a unimodularity property and polynomiality follows from the application of the ellipsoid algorithm [7].

Of course, from an algorithmic perspective, resting the case for formal efficiency on the solvability of linear programs is not at present very insightful. However, we observe that when such a phenomenon has occurred previously, efficient combinatorial algorithms have generally resulted (e.g., matching). This, as much as any other reason has provided the motivation for the present algorithm for the Steiner tree problem. To this extent, it is anticipated that continued research will produce similar results for other interesting problems defined on the class of series-parallel graphs.



(a)



(b)

Figure 1. Sample Problem

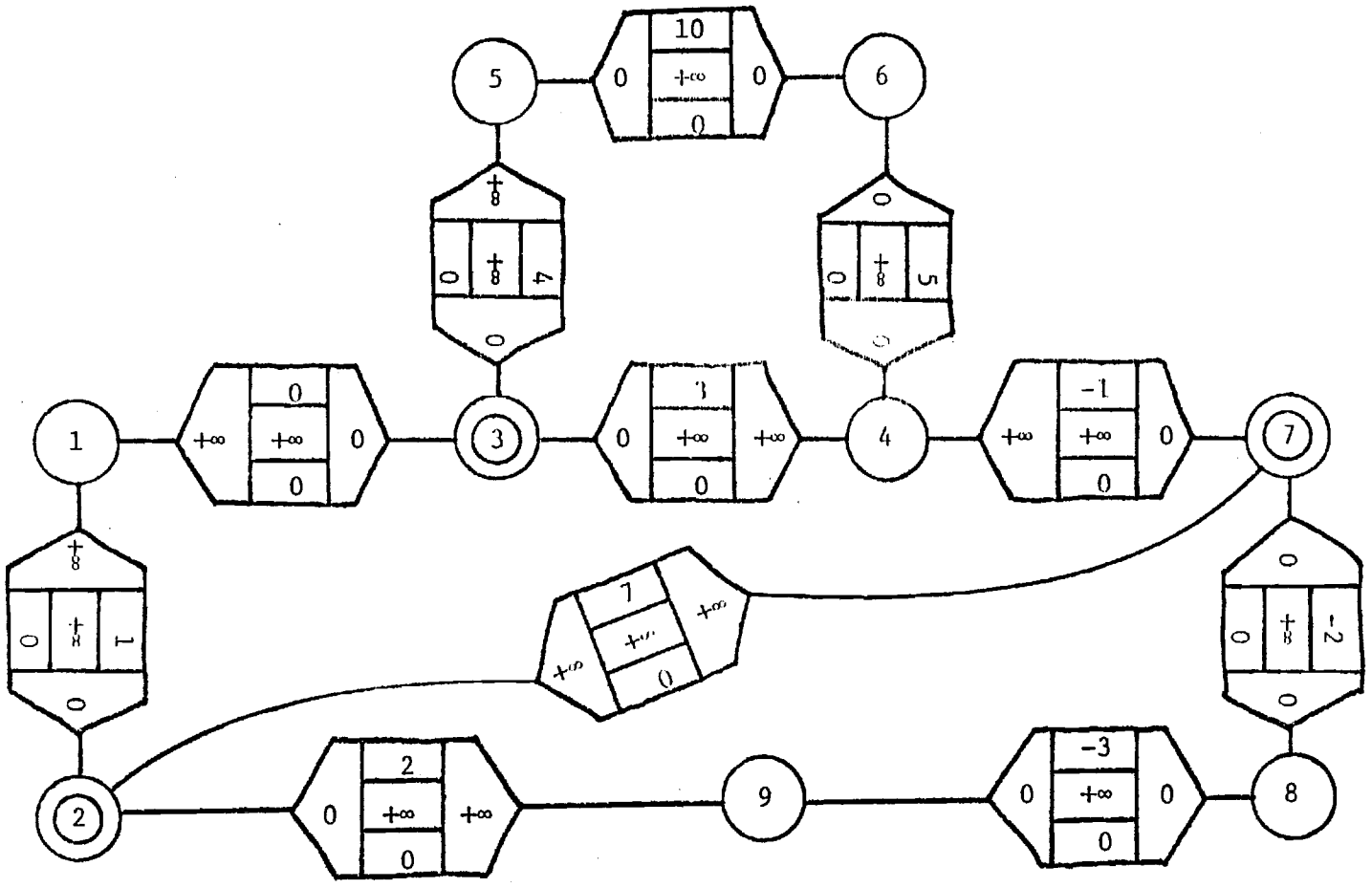


Figure 2. Original Edge Labels for Sample Problem

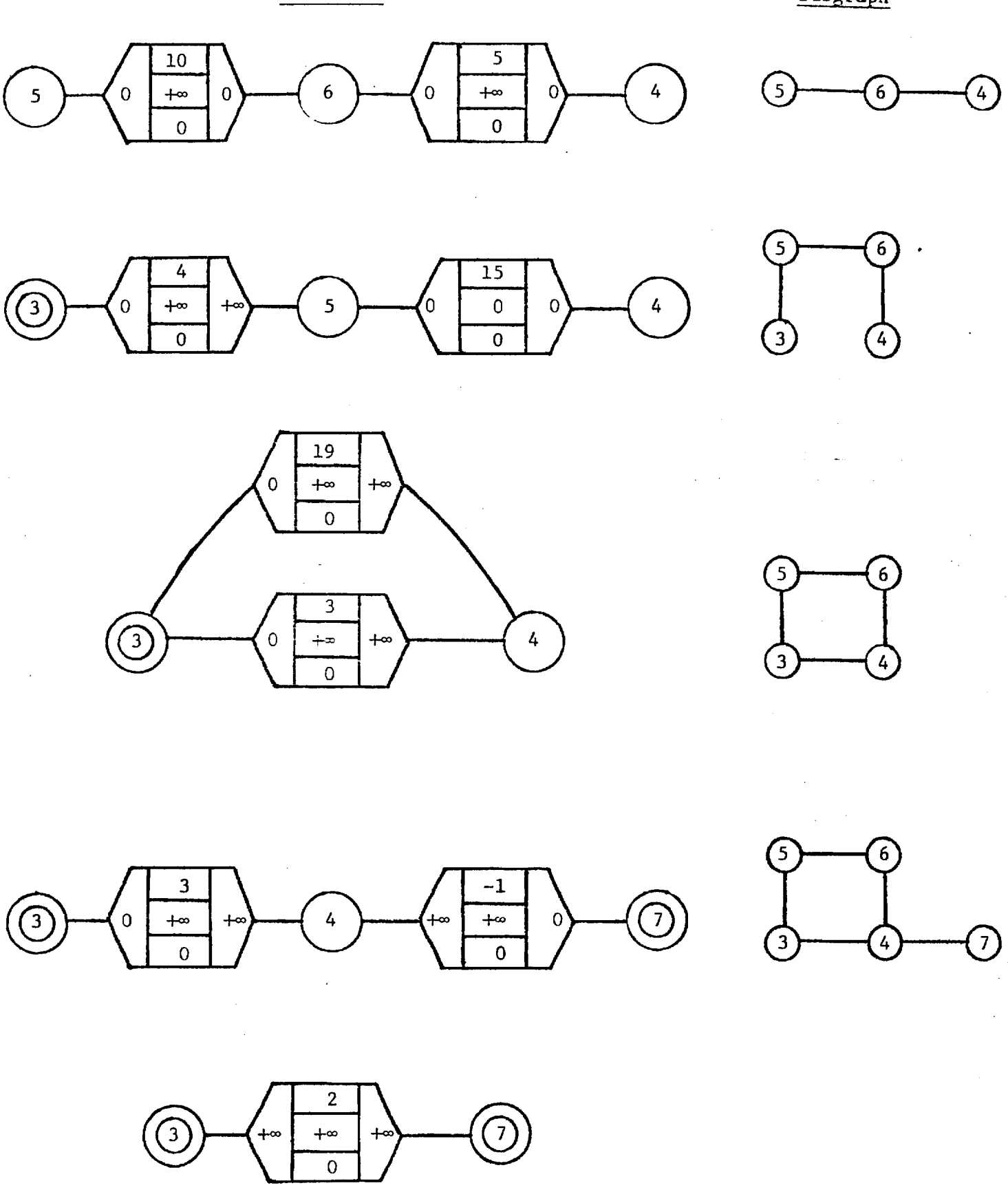


Figure 3. Sequence of Reductions for Sample Problem

Reduction

Subgraph

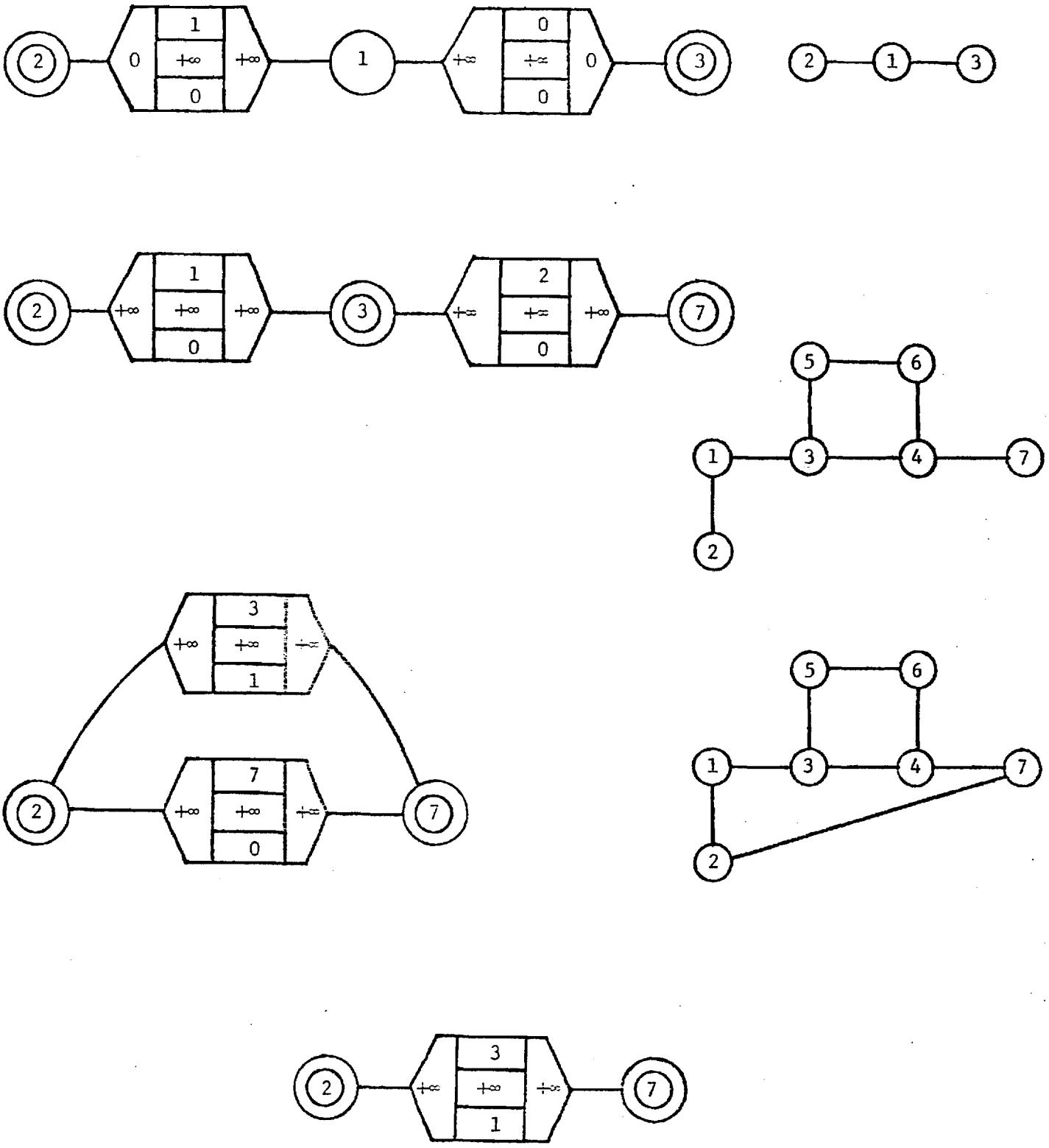
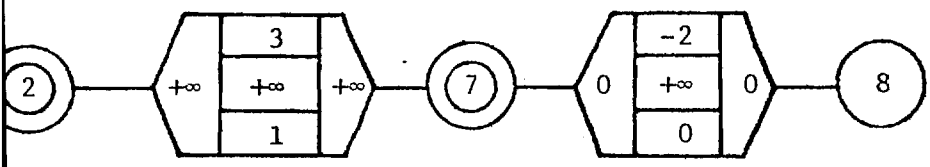
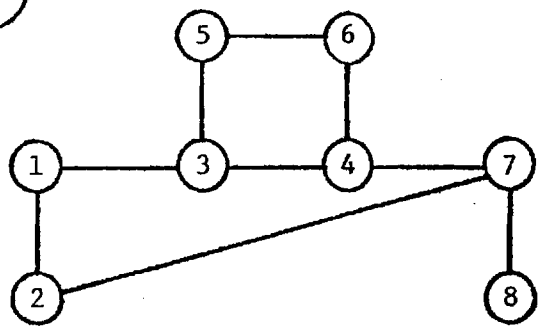
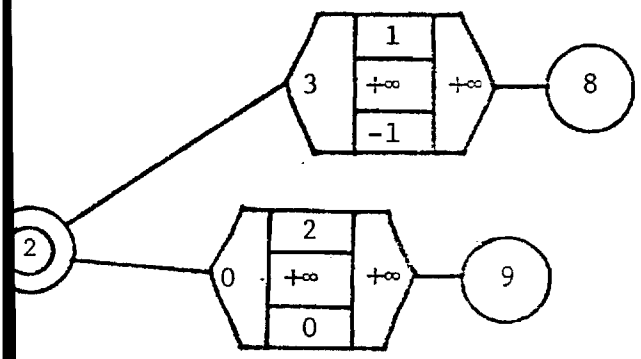
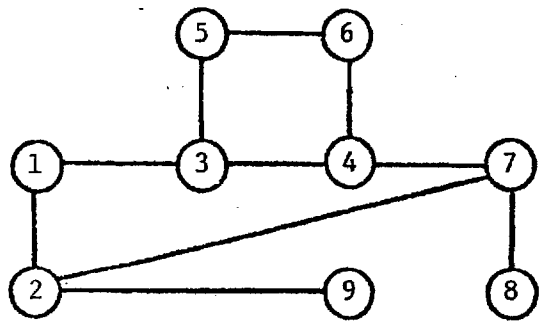


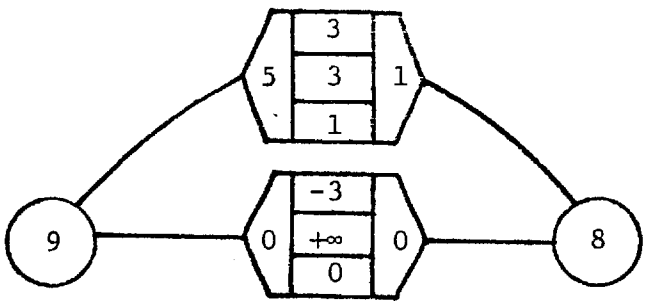
Figure 3. (cont)

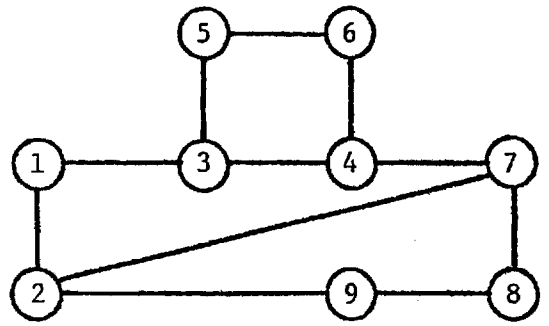












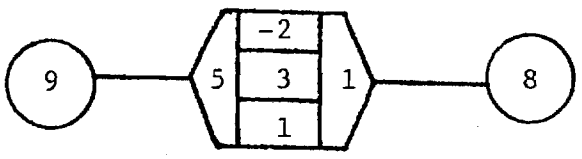


Figure 3. (cont)

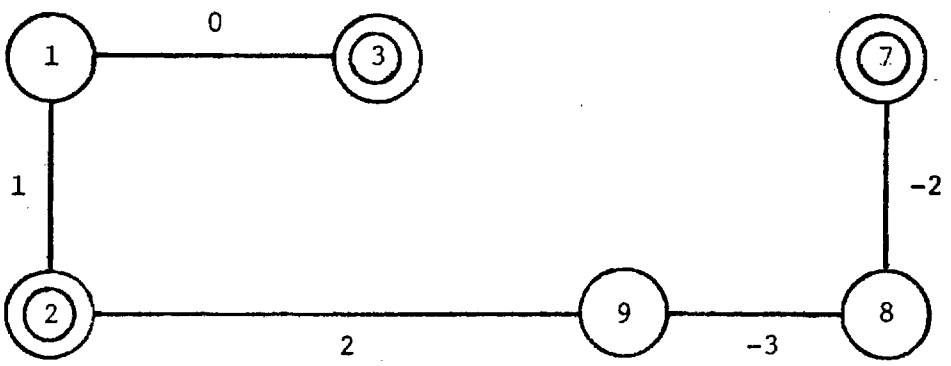


Figure 4. Optimal Steiner Tree for Sample Problem

5. REFERENCES

1. Dreyfus, S. E. and R. A. Wagner, "The Steiner Problem in Graphs," Networks, Vol. 1, No. 3, pp. 195-207, (1972).

2. Duffin, R. J., "Topology of Series-Parallel Networks," J. Math. Anal Appl., Vol. 10, pp. 303-318, (1965).

3. Garey, M. R. and D. S. Johnson, "The Rectilinear Steiner Tree Problem is *NP*-Complete," SIAM J. Appl. Math., Vol. 32, No. 4, pp. 826-834, (1977).

4. Garey, M. R. and D. S. Johnson, Computers and Intractability: A Guide to the Theory of *NP*-Completeness, W. H. Freeman and Co., San Francisco, (1979).

5. Hakimi, S. L., "Steiner's Problem in Graphs and Its Implications," Networks, Vol. 1, No. 2, pp. 113-133, (1971).

6. Karp, R. M., "Reducibility Among Combinatorial Problems," in R. E. Miller and J. W. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, New York, pp. 85-103, (1972).

7. Khachian, L. G., "A Polynomial Algorithm in Linear Programming," Doklady, Akademii Nank, SSSR, 244, (1979).

8. Lawler, E. L., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, (1976).

9. Lawler, E. L., "Sequencing Problems with Series-Parallel Precedence Constraints," Proceedings, Conf. on Combinatorial Optimization, Urbino, Italy, (1978).

10. Rardin, R. L., R. G. Parker, and W. K. Lim, "Some Polynomially Solvable Multi-Commodity Fixed Charge Network Flow Problems," ISyE Report Series J-82-2, Georgia Institute of Technology, Atlanta, GA, (1982).

11. Shore, M. L., L. R. Foulds, and P. B. Gibbons, "An Algorithm for the Steiner Tree Problem on Graphs," Networks, Vol. 12, No. 3. pp. 323-334, (1982).

12. Takamizawa, K., T. Nishizeki, and N. Saito, "Combinatorial Problems on Series-Parallel Graphs," JACM, Vol. 29, No. 3, (1982).

March 1983
Revised: August 1983

A SOLVABLE CASE OF THE MINIMUM WEIGHT EQUIVALENT SUBGRAPH PROBLEM

by

M. B. Richey and R. Gary Parker
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332

R. L. Rardin
School of Industrial Engineering
Purdue University
W. Lafayette, Indiana 47907

This material is based upon work partially supported by the National Science Foundation under Grants ECS-8018954, ECS-8300533 and ECS-8312755.

ABSTRACT

The problem of finding a minimum cardinality subset $\tilde{A} \subseteq A$ such that the subgraph, $G(V,\tilde{A})$, preserves the reachability properties of digraph $G(V,A)$ is well known to be difficult. In this paper, we consider a generalization which seeks a minimum weight subset $\tilde{A}$ satisfying the stated conditions where the weights of arcs in A are assigned arbitrary integer values. A polynomial time algorithm is given for the case where the underlying, undirected graph is series-parallel. Naturally, the stated algorithm subsumes the cardinality case on such graphs as well.

1. INTRODUCTION

In this paper we consider the following problem: given a directed graph, $G(V,A)$, where arcs, $e \in A$ are assigned arbitrary integer weights, $c(e)$, find a minimum weight subset $\tilde{A} \subseteq A$ such that the graph, $G(V,\tilde{A})$ possesses a directed path between vertices u and v in V if an only if $G(V,A)$ does. When $c(e)$ are identical, the problem reduces to one of determining a minimum cardinality subset, and is known accordingly, as the minimum equivalent graph problem (MEGP). Relativising, we shall refer to our problem as the weighted version of the MEGP and hereafter denote it by MWEGP. The corresponding graphs are denoted by MEG and MWEG, respectively.

Relatively little has been done on the MEGP, although three notable papers have appeared: Moyles and Thompson (1969), Hsu (1975) and most recently, Martello and Toth (1982). Regardless, the MEGP is known to be difficult, its intractability having been established in Sahni (1974). Thus, since an algorithm for the MWEGP would trivially solve the MEGP, the former is difficult, as well, for instances defined on arbitrary directed graphs (the problem is uninteresting on undirected graphs since a minimum weight spanning tree provides the solution). However, in what follows we shall demonstrate that when the underlying simple graph of $G(V,A)$ is in the class of graphs referred to as series-parallel, the MWEGP can be solved in polynomial time.

Following, we review some basic notions regarding series-parallel graphs, after which we motivate the development of an algorithm. We then provide a formal statement of the procedure and demonstrate it with a sample problem. After establishing the efficiency of the

algorithm we conclude with some comments regarding other solvable problems on series-parallel graphs.

2. BACKGROUND

Let $G(V,A)$ be a directed graph and denote by $G(V,E)$, its undirected counterpart. That is, $G(V,E)$ is obtained from $G(V,A)$ by simply negecting the orientation of arcs in A. We then have that $G(V,E)$ is series-parallel if and only if it can be reduced to an edge by the sequential application of the following elementary operations:

(i) Series-reduction: replace any degree-2 vertex, k, and its incident edges (or pseudo-edges), e and f, connecting k to vertices i and $j \neq i$, by a pseudo-edge, g, incident to i and j.

(ii) Parallel-reduction: replace any two edges (either or both of which may be pseudo-edges), e and f, both incident to vertices i and j, by a pseudo-edge, g, incident to i and j.

(iii) Jackknife-reduction: replace any degree-1 vertex, k, its incident edge e=(j,k), and any other edge, f=(i,j), incident to e, by a pseudo-edge, g=(i,j).

Alternately, Duffin (1965) has given the following characterization of series-parallel graphs.

Theorem 1: A loopless, undirected graph is series-parallel if and only if it possesses no subgraph homeomorphic to K_4 (the complete graph on 4 vertices).

There are also other specifications of series-parallel graphs whose equivalence with either of the two above is established in Rardin, Parker and Wagner (1982). Regardless, it is obvious that series-parallel graphs form a subset of planar graphs since K_4 itself is planar. It is also

worth observing that 2-connected graphs free of K_4 homeomorphs are reducible to an edge by invoking only the series and parallel reductions.

The latter observation above is important since it allows a slight simplification for the MWEGP. Clearly, any degree-1 vertex in a graph (either the original graph or one which has been reduced) implies the existence of a cut-vertex. Hence, if any pair of vertices (neither of which is the cut-vertex) in the two blocks induced by the cut-vertex are connected by a path, the path must include the cut-vertex. This, in turn, implies that there is no loss of generality if we consider only instances of the MWEGP which are defined on 2-connected graphs. To be consistent, we simply modify Theorem 1 accordingly and eliminate reduction operation (iii).

Of course, there is discretion in how the series and parallel operations are applied on a graph. As it turns out, however, this application can be arbitrary. We have

Theorem 2: If $G(V,E)$ is a 2-connected, series-parallel graph, then any admissable sequence of operations (i) and (ii) will reduce G to a single edge.

Proof: See Richey, et.al. (1982).

In Figure 1, we illustate the reduction process. Here, the directed graph, $G(V,A)$ is converted to its underlying, undirected counterpart, $G(V,E)$ after which reductions (i) and (ii) are applied, culminating with a single edge.

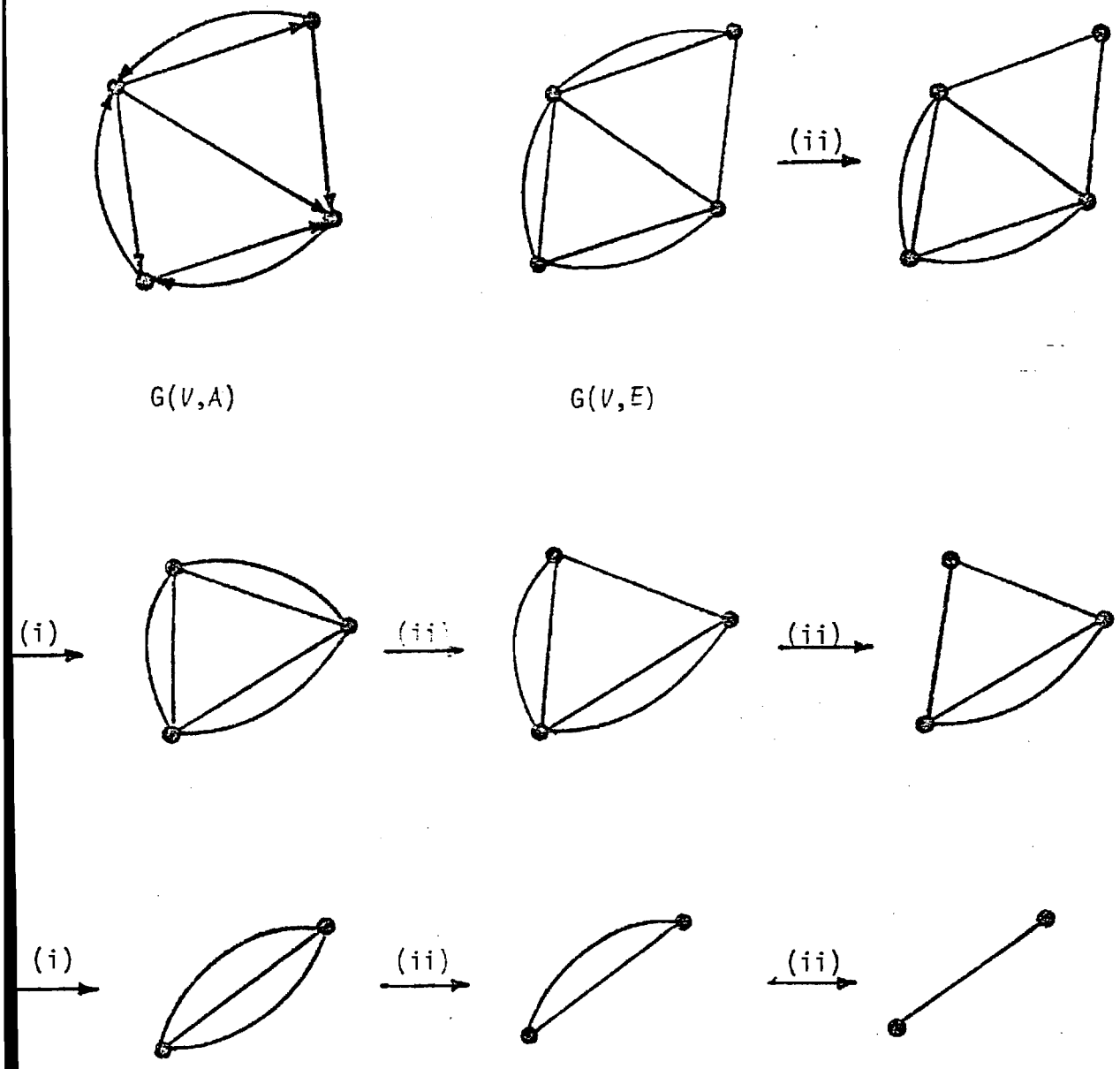


Figure 1. Series and Parallel Reductions

3. DEVELOPMENT OF THE ALGORITHM

3.1 General Concepts

During the reduction process, pseudo-edges are formed by the series and parallel reduction operations. Each pseudo-edge represents a subgraph of the original graph. Of course, ascertaining the relationship between the MWEGs of these subgraphs and the MWEGs of their series and parallel combinations is at the heart of the ensuing algorithm.

Suppose we let $G_r(V_r,E_r)$ denote the graph resulting from the application of a sequence of series and parallel reductions to the underlying, undirected counterpart of a directed graph, $G(V,A)$. In addition, define $G[e]$ to be the subgraph of $G(V,A)$ with vertex and edge sets $V[e]$ and $E[e]$ respectively, corresponding to $e \in E_r$.

Now, a MWEG of $G[e]$ must possess a path between any two vertices of $G[e]$ if the vertices are connected by a path in $G[e]$. In particular, consider paths between a terminal of $G[e]$ (i.e. an element of $V[e] \cap V_r$) and some other vertex in $V[e]$. Clearly, any path between $i \in V[e]$ and $j \in V \setminus V[e]$ must use at least one terminal of $G[e]$. Thus, if i and $j \neq i$ are elements of $V[e]$, then any path between them which is not contained in $G[e]$ must pass through both terminals of $G[e]$. This is so since a path passing through one terminal twice can be considered as two separate pieces, the cycle exterior to $G[e]$ and the path interior to $G[e]$. We are lead to the following lemma:

Lemma 3: If the values of the minimum weight subgraphs of $G[e]$ are known with respect to the following properties for every $e \in E_r$, then the value of the MWEG of $G(V,A)$ can be determined.

(i): Subgraph equivalent to G[e], denoted by (MWEG)

(ii): Subgraph equivalent to G[e] which has a path from terminal i to terminal j (both of G[e]), denoted by (→)

(iii): Subgraph equivalent to G[e] which has a path from terminal j to terminal i (both of G[e]), denoted by (←)

(iv): Subgraph equivalent to G[e] which has a path from i to j and a path from j to i, denoted by (↔)

(v): Subgraph which would be equivalent to G[e] if a path from i to j were added to the subgraph, denoted by (↛)

(vi): Subgraph which would be equivalent to G[e] if a path from j to i were added to the subgraph, denoted by (↚)

(vii): Subgraph which would be equivalent to G[e] if a path from i to j and a path from j to i were added to the subgraph, denoted by (↮)

(viii): Subgraph which has a path from i to j and would be equivalent to G[e] if a path from j to i were added, denoted by (⇄)

(ix): Subgraph which has a path from j to i and would be equivalent to G[e] if a path from i to j were added, denoted by (⇆)

Note: In (v)-(ix), subgraphs which "would be equivalent to G[e] if ..." include subgraphs which are equivalent to G[e] without the specified path(s).

Proof: Consider vertices $i \in V[e_1]$ and $j \in V[e_\ell]$ such that there exists a path from i to j in $G(V,A)$ which passes through edges (in order) $e_1, e_2, \ldots, e_\ell \in E_r$ where $\ell > 1$. Our proof will be by induction on ℓ.

Clearly, the lemma is true when $\ell = 1$ for by the discussion proceeding the lemma, each path between i and j must be entirely in $G[e_1]$ or else consist of arcs in $E[e_1]$ added to a simple path between the terminals of e_1. Since all possible combinations of paths between the terminals are included in the nine specified subgraphs, every possible subgraph of $G[e_1]$ which could include a path from i to j in the MWEG of

$G(V,A)$ is accounted for.

Now, assume the lemma to be true for $\ell = k$. Then for $\ell = k + 1$, let t be the terminal vertex shared by e_k and e_{k+1}. Since $t \in G[e_k]$, all possible paths between i and t which could be in the MWEG of $G(V,A)$ have been considered. In addition, since t and j are in $G[e_{k+1}]$, all possible paths between t and j have been considered as well. Further, this is true for all possible terminals t, and hence all possible paths between the stated vertices i and j are accounted for by the nine specified subgraphs. This completes the proof.

As it turns out, it is convenient to think of nine subgraphs not as separate cases but rather in terms of their interrelationships. That is, a given case can be viewed in terms of those cases which are restrictions of it. This leads to the hierarchy in Figure 2. Interpreting, a given case denoted by node i includes another, i' if there is a directed path from node i to node i' in the figure.

Of course, no minor question at this point is that regarding the actual calculation of the subgraph labels specified in Lemma 3. That is, how are such values determined for a pseudo-edge from its constituent real edges?

Let L[e] be a nine-tuple for pseudo- edge e with elements related as

$$(\leftrightarrow, \rightarrow, \leftarrow, \text{MWEG}, \overleftarrow{\not\rightarrow}, \overrightarrow{\not\leftarrow}, \not\rightarrow, \not\leftarrow, \not\leftrightarrow)$$

and denoted by $\ell(e,*)$ a given value in the nine-tuple, i. e. $* \in \{\leftrightarrow, \rightarrow, \ldots, \not\leftrightarrow\}$. Then the next two lemmas show how pseudo- edges interact.

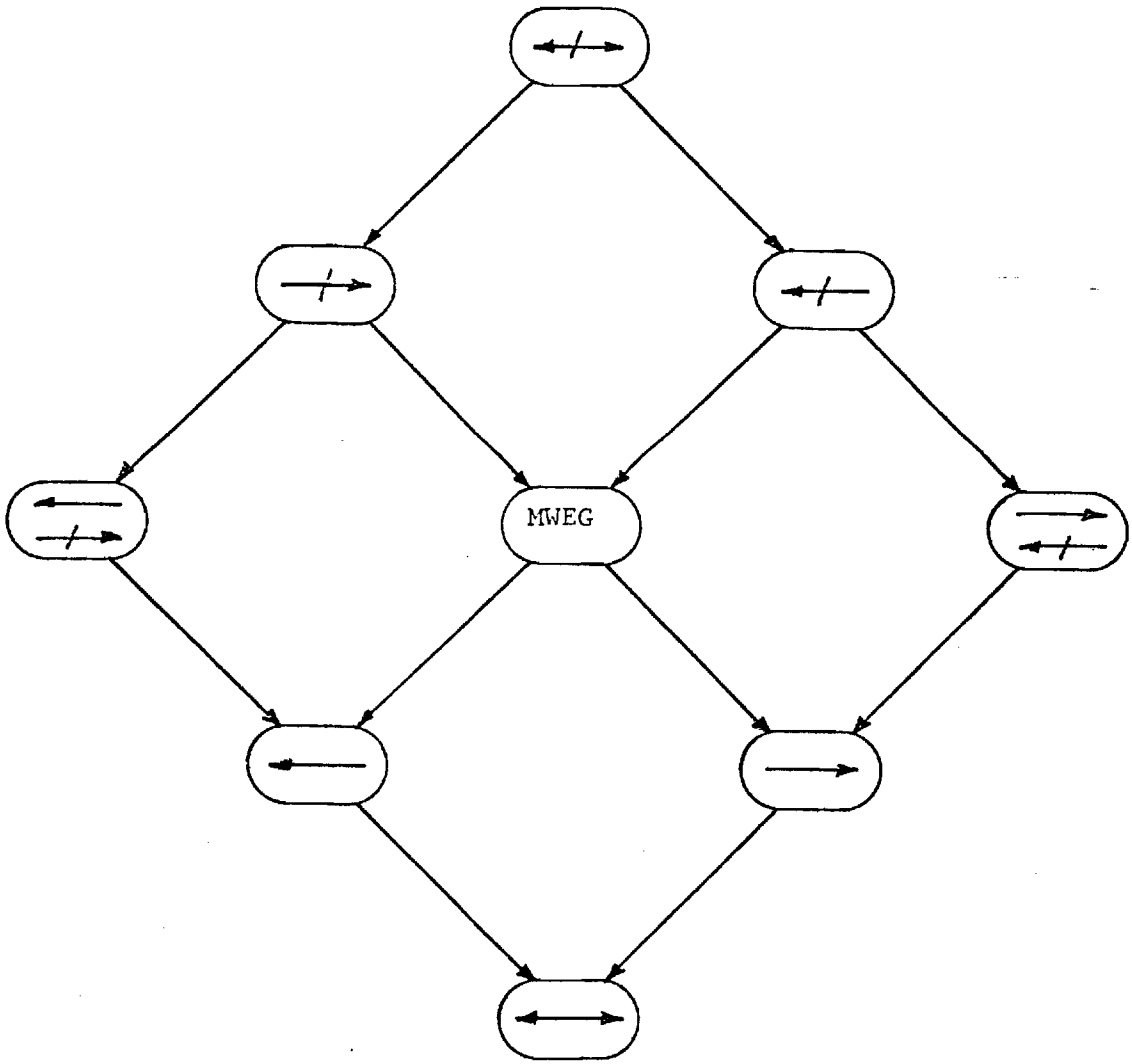


Figure 2. Hiearchy of Subgraph Labels

Lemma 4: Let pseudo- edges e and f be series-reduced, forming pseudo-edge g. Then

$$\ell(g,\leftrightarrow) = \ell(e,\leftrightarrow) + \ell(f,\leftrightarrow)$$

$$\ell(g,\rightarrow) = \ell(e,\rightarrow) + \ell(f,\rightarrow)$$

$$\ell(g,\leftarrow) = \ell(e,\leftarrow) + \ell(f,\leftarrow)$$

$$\ell(g,\mathrm{MWEG}) = \ell(e,\mathrm{MWEG}) + \ell(f,\mathrm{MWEG})$$

$$\ell(g,\overrightarrow{\not\leftarrow}) = \ell(e,\overrightarrow{\not\leftarrow}) + \ell(f,\overrightarrow{\not\leftarrow})$$

$$\ell(g,\overleftarrow{\not\rightarrow}) = \ell(e,\overleftarrow{\not\rightarrow}) + \ell(f,\overleftarrow{\not\rightarrow})$$

$$\ell(g,\not\leftrightarrow) = \min \begin{cases} \ell(e,\leftarrow) + \ell(f,\not\leftrightarrow) \\ \ell(e,\not\leftrightarrow) + \ell(f,\leftarrow) \\ \ell(e,\overleftarrow{\not\rightarrow}) + \ell(f,\overleftarrow{\not\rightarrow}) \\ \ell(g,\mathrm{MWEG}) \end{cases}$$

$$\ell(g,\not\leftarrow) = \min \begin{cases} \ell(e,\rightarrow) + \ell(f,\not\leftarrow) \\ \ell(e,\not\leftarrow) + \ell(f,\rightarrow) \\ \ell(e,\overrightarrow{\not\leftarrow}) + \ell(f,\overrightarrow{\not\leftarrow}) \\ \ell(g,\mathrm{MWEG}) \end{cases}$$

$$\ell(g,\not\leftrightarrow) = \min \begin{cases} \ell(e,\leftrightarrow) + \ell(f,\not\leftrightarrow) \\ \ell(e,\not\leftrightarrow) + \ell(f,\leftrightarrow) \\ \ell(g,\not\rightarrow) \\ \ell(g,\not\leftarrow) \end{cases}$$

Proof: In Table 1, the potential resulting graph from each of the 81 label combinations are shown. Observe that crossed-out cells indicate subgraphs which cannot become part of the MWEG of the original graph due to Lemma 3. Cells marked by asterisks indicate subgraphs which dominate

the others in Figure 2. These combinations have simply been re-organized to produce the list given by the lemma.

The reader may note that the series combinations preserve symmetry. For parallel reduction, we have

Lemma 5: Let pseudo- edges e and f be parallel-reduced, forming pseudo-edge g. Then

$$\ell(g,\leftrightarrow) = \min \begin{cases} \ell(e,\leftrightarrow) + \ell(f,\not\leftrightarrow) \\ \ell(e,\not\leftrightarrow) + \ell(f,\leftrightarrow) \\ \ell(e,\overrightarrow{\not\leftrightarrow}) + \ell(f,\overleftarrow{\not\leftrightarrow}) \\ \ell(e,\overleftarrow{\not\leftrightarrow}) + \ell(r,\overrightarrow{\not\leftrightarrow}) \end{cases}$$

$$\ell(g,\rightarrow) = \min \begin{cases} \ell(e,\rightarrow) + \ell(f,\not\rightarrow) \\ \ell(e,\not\rightarrow) + \ell(f,\rightarrow) \\ \ell(g,\leftrightarrow) \end{cases}$$

$$\ell(g,\leftarrow) = \min \begin{cases} \ell(e,\leftarrow) + \ell(f,\not\leftarrow) \\ \ell(e,\not\leftarrow) + \ell(f,\leftarrow) \\ \ell(g,\leftrightarrow) \end{cases}$$

$$\ell(g,\text{MWEG}) = \min \begin{cases} \ell(e,\text{MWEG}) + \ell(f,\text{MWEG}) \\ \ell(g,\rightarrow) \\ \ell(g,\leftarrow) \end{cases}$$

$$\ell(g,\overrightarrow{\not\leftrightarrow}) = \min \begin{cases} \ell(e,\overrightarrow{\not\leftrightarrow}) + \ell(f,\not\leftrightarrow) \\ \ell(e,\not\leftrightarrow) + \ell(f,\overrightarrow{\not\leftrightarrow}) \end{cases}$$

$$\ell(g,\overleftarrow{\not\leftrightarrow}) = \min \begin{cases} \ell(e,\overleftarrow{\not\leftrightarrow}) + \ell(f,\not\leftrightarrow) \\ \ell(e,\not\leftrightarrow) + \ell(f,\overleftarrow{\not\leftrightarrow}) \end{cases}$$

$$\ell(g,\not\mapsto) = \min \begin{cases} \ell(e,\not\mapsto) + \ell(f,\not\mapsto) \\ \ell(g,\overrightarrow{\not\mapsto}) \end{cases}$$

$$\ell(g,\not\leftarrow) = \min \begin{cases} \ell(e,g\not\leftarrow) + \ell(f,\not\leftarrow) \\ \ell(g,\overrightarrow{\not\leftarrow}) \end{cases}$$

$$\ell(g,\not\leftrightarrow) = \ell(e,\not\leftrightarrow) + \ell(f,\not\leftrightarrow)$$

Proof: Employing Table 2 rather than Table 1, the proof follows in analogous fashion to that of Lemma 4. We leave the details to the reader.

3.2 The Algorithm

We are now in a position to state a computational procedure. We present the scheme in step-by-step fashion.

Step 0: Initialization. For the given instance, $G(V,A)$ let $G_r(V_r,E_r)$ be the associated indirected graph. Denote as c[e] the weight of arc e corresponding to edge $e \in E_r$ and let vertex i be the left vertex of e and j, the right vertex. Initialize L[e] as follows:

$$\ell(e,\leftrightarrow) = +\infty$$

$$\ell(e,\rightarrow) = \begin{cases} c(e) \text{ if } e = \langle i,j\rangle \\ +\infty \text{ if } e = \langle j,i\rangle \end{cases}$$

$$\ell(e,\leftarrow) = \begin{cases} +\infty \text{ if } e = \langle i,j\rangle \\ c(e) \text{ if } e = \langle j,i\rangle \end{cases}$$

$$\ell(e,\text{MWEG}) = c(e)$$

$$\ell(e,\overrightarrow{\not\leftarrow}) = \ell(e,\rightarrow)$$

$$\ell(e,\overrightarrow{\not\mapsto}) = \ell(e,\leftarrow)$$

$$\ell(e,\not\rightarrow) = \begin{cases} \min(c(e),0) & \text{if } e = \langle i,j\rangle \\ c(e) & \text{if } e = \langle j,i\rangle \end{cases}$$

$$\ell(e,\not\leftarrow) = \begin{cases} c(e) & \text{if } e = \langle i,j\rangle \\ \min(c(e),0) & \text{if } e = \langle j,i\rangle \end{cases}$$

$$\ell(e,\not\leftrightarrow) = \min(c(e),0)$$

Note that $\langle i,j\rangle$ denotes the ordered pair defining an element of A.

Step 1: Reduction. Beginning with the initial G_r and referring to Lemmas 4 and 5, perform series and parallel reductions arbitrarily, updating G_r after each. When G_r can be reduced no further, go to Step 2.

Step 2: Stopping. If G_r is a single pseudo- edge, stop; the problem solution is at hand. The MWEG of G can be obtained by backtracking through the sequence of reductions which produced G_r. If G_r is not a single edge, then the underlying undirected graph of $G(V,A)$ is not series-parallel.

Table 1. Series Reduction

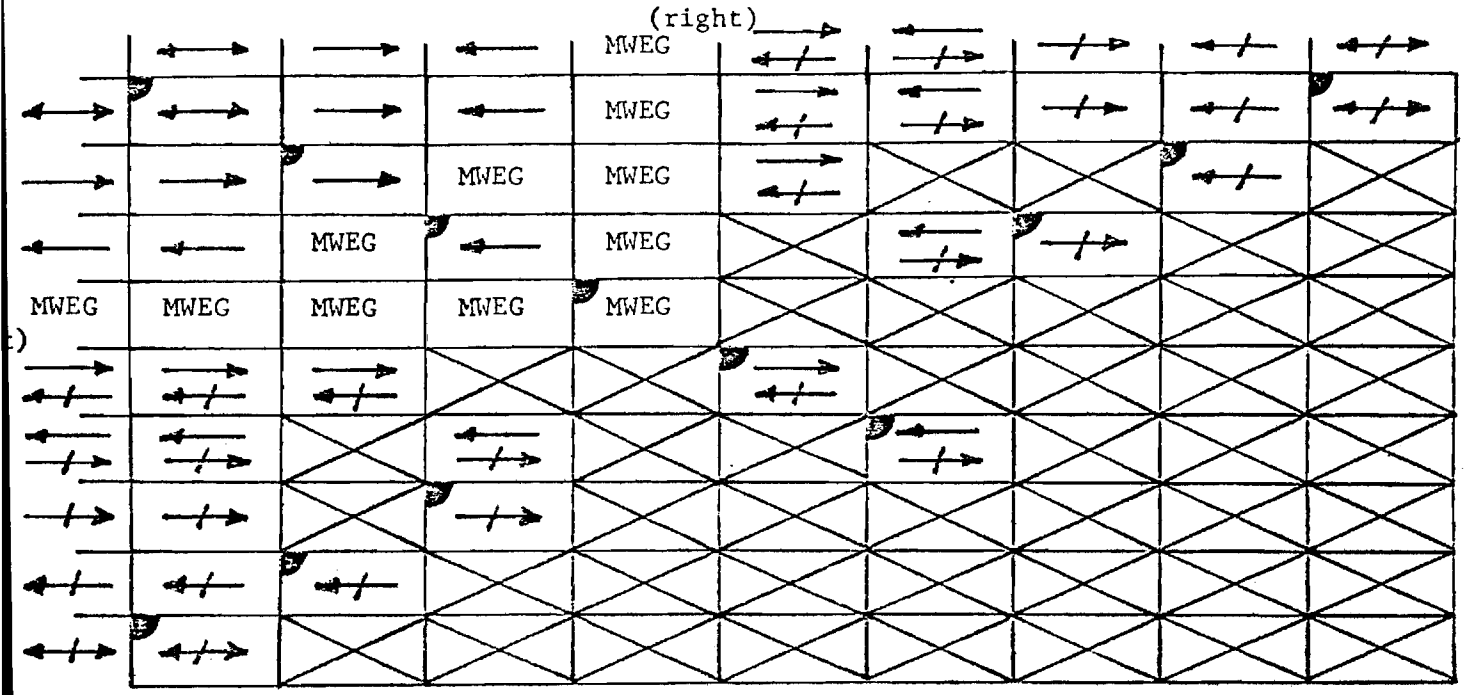
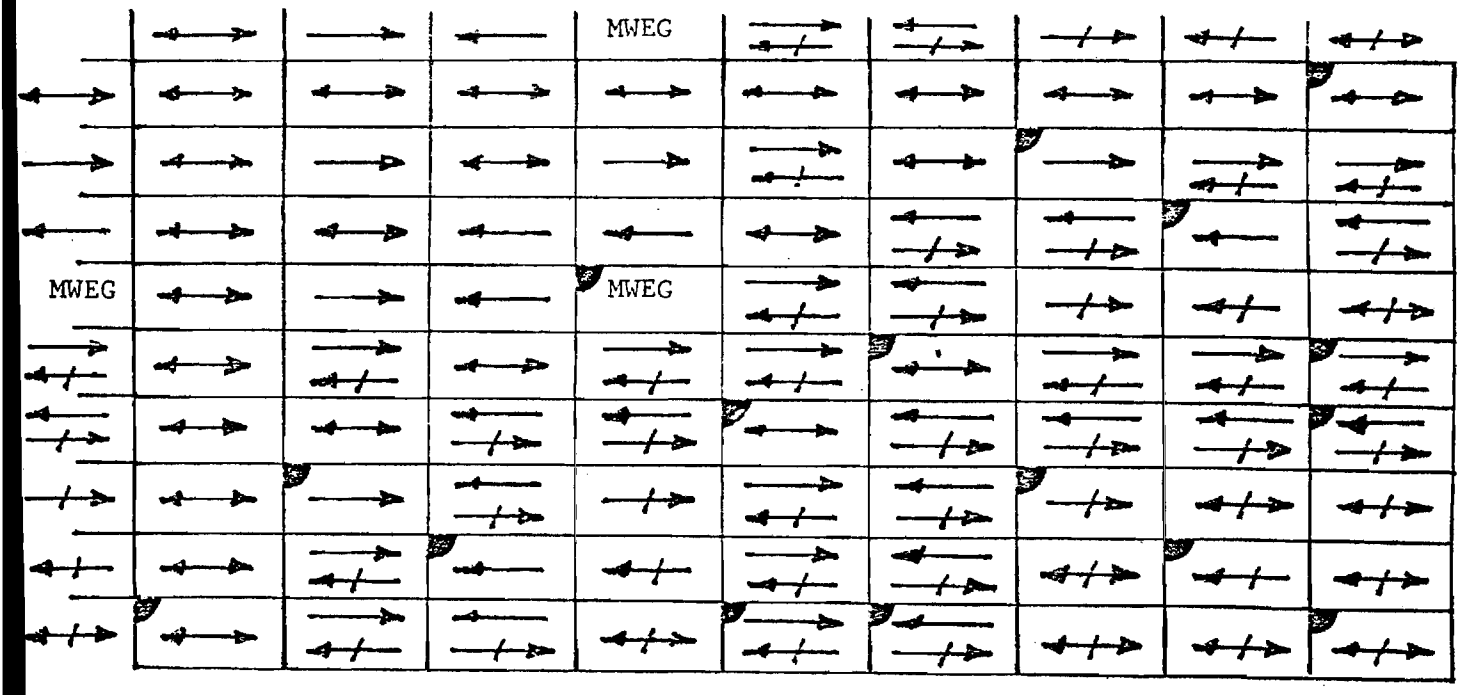


Table 2. Parallel Reduction



3.3 An Example

In order to demonstrate the algorithm, consider the graph shown in Figure 3. Weights are specified directly on the arcs. Initializing we have:

L[1,2]=(∞,-1,∞,-1,-1,∞,-1,-1,-1)

L[1,3]=(∞,∞,-3,-3,∞,-3,-3,-3,-3)

L[1,4]=(∞,4,∞,4,4,∞,0,4,0)

L[1,5]=(∞,∞,6,6,∞,6,6,0,0)

L[2,3]=(∞,2,∞,2,2,∞,0,2,0)

L[3,4]=(∞,∞,5,5,∞,5,5,0,0)

L[3,7]=(∞,7,∞,7,7,∞,0,7,0)

L[5,6]=(∞,∞,1,1,∞,1,1,0,0)

L[5,8]=(∞,3,∞,3,3,∞,0,3,0)

L[6,7]=(∞,∞,2,2,∞,2,2,0,0)

L[7,9]=(∞,∞,-4,-4,∞,-4,-4,-4,-4)

L[7,10]=(∞,∞,5,5,∞,5,5,0,0)

L[8,9]=(∞,-6,∞,-6,-6,∞,-6,-6,-6)

L[9,8]=(∞,∞,7,7,∞,7,7,0,0)

L[8,11]=(∞,2,∞,2,2,∞,0,2,0)

L[11,8]=(∞,∞,9,9,∞,9,9,0,0)

L[9,10]=(∞,∞,1,1,∞,1,1,0,0)

L[9,11]=(∞,3,∞,3,3,∞,0,3,0)

L[9,12]=(∞,∞,3,3,∞,3,3,0,0)

L[11,12]=(∞,∞,1,1,∞,1,1,0,0)

The entire computation can be summarized by Table 3 where specific choices for reduction are shown in the first column with the pseudo-edge creation and label computation in the second and third. Observe that we denote series and parallel reduction as S and P respectively. The backtracking process is depicted by the boxed label components and the resulting MWEG is shown in Figure 4.

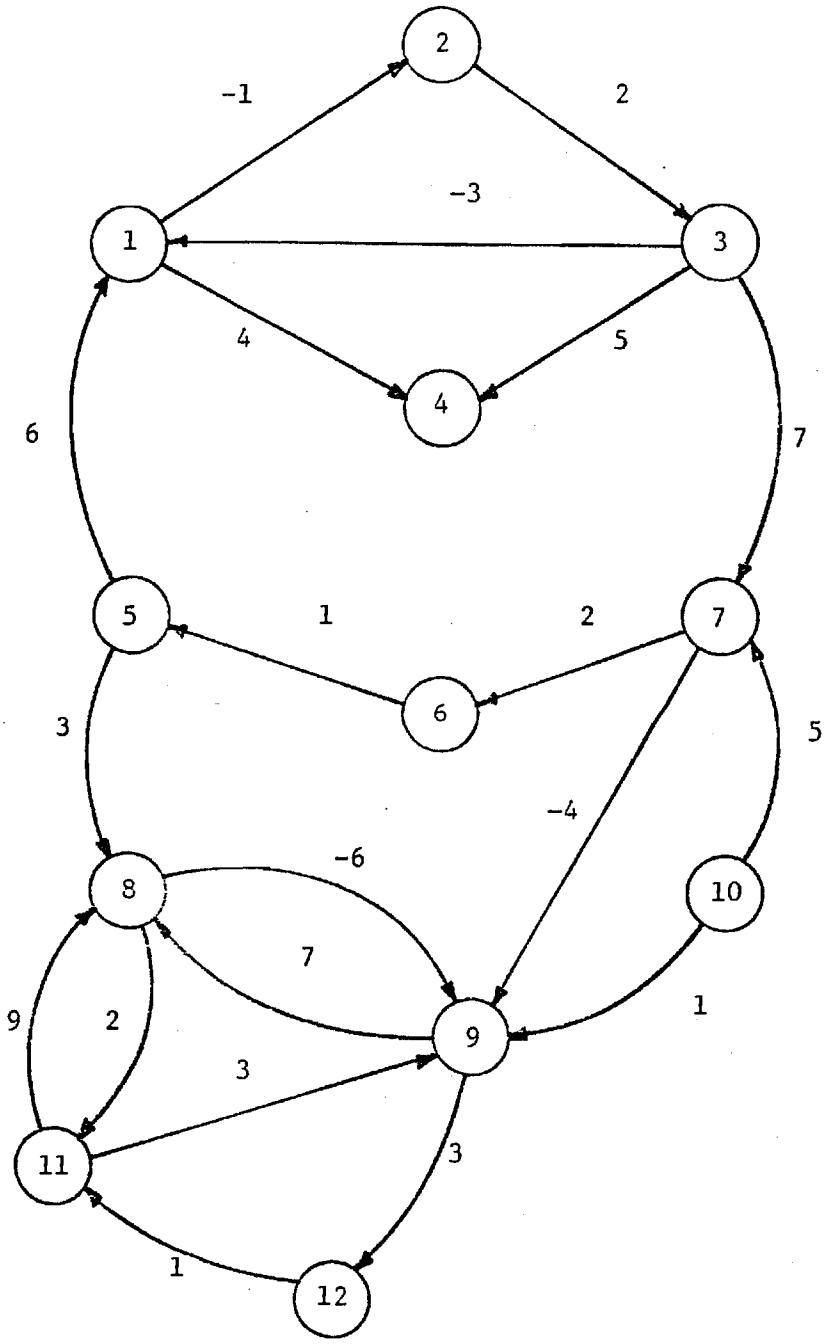
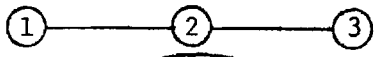
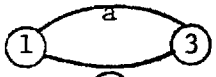
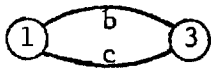
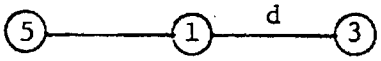
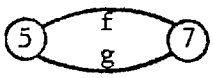
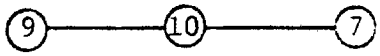
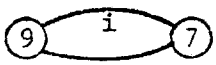
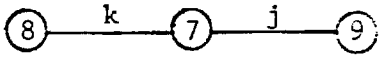
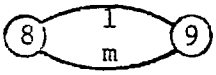
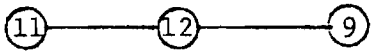
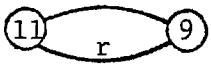
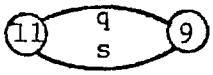


Figure 3 Graph of Example

Table 3. Summary of Computation for Example Problem

(1)		S	L[a]=(∞,1,∞,1,[1],∞,1,1,1)
(2)		P	L[b]=([-2],-2,-2,-2,-2,-2,-2,-2,-2)
(3)		S	L[c]=(∞,∞,∞,9,∞,∞,5,4,[4])
(4)		P	L[d]=(2,[2],2,2,2,2,2,2,2)
(5)		S	L[e]=(∞,[8],∞,8,8,∞,2,8,2)
(6)		S	L[f]=(∞,[15],∞,15,15,∞,15,15,15)
(7)		S	L[g]=(∞,∞,3,3,∞,3,[3],3,3)
(8)		P	L[h]=(18,[18],18,18,18,18,18,18,18)
(9)		S	L[i]=(∞,∞,∞,6,∞,∞,1,[5],1)
(10)		P	L[j]=(∞,∞,[1],1,∞,-3,-3,1,-3)
(11)		S	L[k]=(∞,∞,21,21,∞,21,21,[18],18)
(12)		S	L[l]=(∞,∞,∞,22,∞,∞,22,19,[19])
(13)		P	L[m]=(1,1,1,1,[-6],1,1,-6,-6)
(14)		P	L[n]=(20,20,20,20,[13],20,20,13,13)
(15)		P	L[p]=(11,11,11,11,2,[9],9,2,0)
(16)		S	L[q]=(31,31,31,31,[22],22,22,22,20)
(17)		S	L[r]=(∞,∞,4,4,∞,[4],4,4,4)
(18)		P	L[s]=(7,7,7,7,7,[4],4,7,4)
(19)		P	L[t]=(26,26,26,[26],26,24,24,26,24)

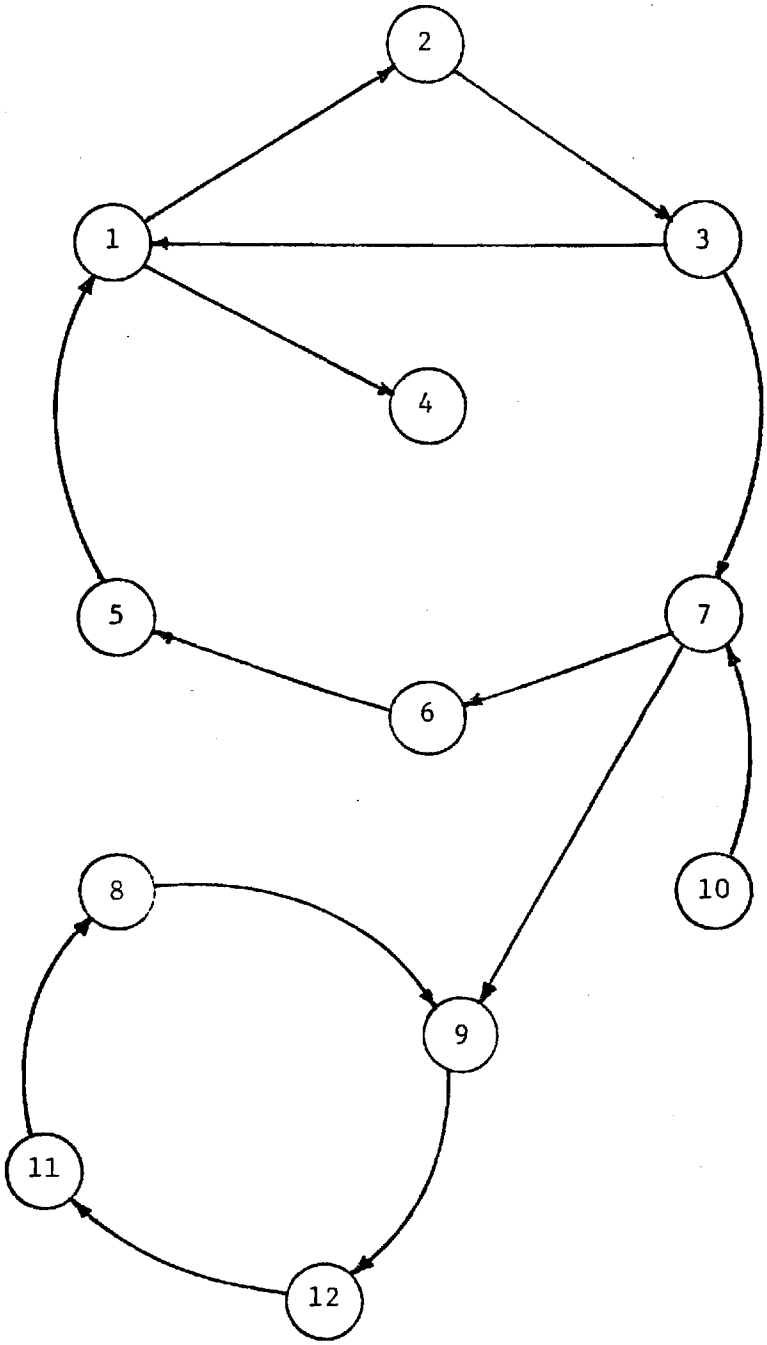


Figure 4 MWEG of Example

3.4 Discussion

Of course, crucial issues regarding the algorithm presented are its correctness as well as its computational requirements. Accordingly, the following theorem summarizes:

Theorem 6: The MWEG algorithm will, in polynomial time, correctly produce a desired subgraph or stop with the conclusion that the input (undirected) graph is not series-parallel.

Proof: First consider stopping and suppose that step 2 is reached with G_r not an edge. Then the minimum vertex degree in G_r is three and thus, G_r must possess a subgraph homemorphic to K_4 [e.g. Dirac (1952), Richey, et. al. (1982)]. But this must mean that the original graph $G(V,E)$ possesses a subgraph homeomorphic to K_4 and by Theorem 1 is not series-parallel. Regardless, step 2 must be reached after polynomially many series and/or parallel reductions which is immediate since simply testing if an arbitrary graph is series-parallel can be done in $O(|V| + |E|)$ steps as shown in Rardin, et. al. (1982).

A correct structure must be produced by the algorithm since in the proofs of Lemmas 3-5, all possibilities for subgraph construction via the reduction process are accounted for. In addition, the pseudo-edge label updates require computation independent of instance size leaving in tact the stated polynomiality of the procedure.

We observe that alternative statements of polynomial time series-parallel testing are available in the literature in addition to that alluded to in the proof above. Among these are Liu and Geldmacher (1980) and Takamizawa, Nishizeki and Saito (1982).

4. SUMMARY

In this paper, we have confined our interest to a specific problem on series-parallel graphs. Other results have emerged in this regard as well, notable among which are Takamizawa, et.al. (1982) and Wald and Colbourn (1983). In the latter, an algorithm is presented for the Steiner tree problem- a result also obtained (independently) by the present authors (see Rardin,et.al. (1982)). Regardless, the impetus for our specific development here pertaining to the MWEGP is based largely on the requirement that instances be defined on directed graphs. That is, the dimension of directionality inherent in the problem clearly gives rise to a degree of complication which may not be present in many problems defined on undirected (series-parallel) graphs.

Of course, it is worthwhile to think of specific algorithmic strategies for other problems which might also be resolvable on series-parallel graphs. To this extent, it is also interesting to think of ones which might be intractable (yet solvable on trees, say). Even so, a fundamental question which is worthy of pursuit pertains to the gap between series-parallel and planar graphs. It appears that the former class of graphs is rich in terms of interesting problems which are amenable to efficient solution procedures while for the latter, many problems are known to be difficult. An investigation of the territory between these two classes would seem to be an interesting undertaking.

REFERENCES

1. Dirac, G. A. (1982), "A Property of 4-Chromatic Graphs and Some Remarks on Critical Graphs", J. London Math. Soc., 27, 85-92.

2. Duffin, R.J. (1965), "Topology of Series-Parallel Networks", J. Math. Anal. Appl., 10, 303-318.

3. Hsu, H.T. (1975), "An Algorithm for Finding a Minimum Equivalent Graph of a Digraph", J. of the ACM, 22, 11-16.

4. Liu, P. C. and R. C. Geldmacher,(1980),"An O(max(m,n)) Algorithm for Finding a Subgraph Homeomorphic to K_4" Proc. 11th Southeastern Conf, on Combinatorics, Graph Theory, and Computing, 597-609.

5. Martello, S. and P. Toth (1982),"Finding a Minimum Equivalent Graph of a Digraph", Networks, 12, 89-100.

6. Moyles, D.M. and G.L. Thompson (1969), "An Algorithm for Finding a Minimum Equivalent Graph of a Digraph," J. of the ACM, 16, 455-460.

7. Rardin, R.L., R.G. Parker, and M.B. Richey (1982), "A Polynomial Algorithm for a Class of Steiner Tree Problems on Graphs," ISyE Report Series J-82-5, Georgia Tech, August.

8. Rardin, R.L., R.G. Parker, and D.K. Wagner, (1982), "Definitions, Properties and Algorithms for Detecting Series-Parallel Graphs," Technical Report, Department of Industrial Engineering, Purdue University, W. Lafayette, IND.

9. Richey, M.B., R.G. Parker, and R.L. Rardin, (1982) "On a Class of Graphs Possessing at Most One Hamiltonian Cycle", ISyE Report Series J-82-11, Georgia Tech., November.

10. Sahni, S., (1974), "Computationally Related Problems," SIAM J. Computing, 3, 262-279.

11. Takamizawa, K., T. Nishizeki and N. Saito (1982), "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs", J. of the ACM, 29, 623-641.

12. Wald, J.A. and C.J. Colbourn (1983), "Steiner Trees, Partial 2-Trees, and Minimum IFI Networks", Networks, 13, 159-167.

Industrial and Systems Engineering
Report Series J-82-11
November, 1982

ON A CLASS OF GRAPHS HAVING AT MOST ONE HAMILTONIAN CYCLE

by

M. B. Richey†
R. Gary Parker†
and
R. L. Rardin†

J-82-11

†School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

This material is based in part upon work partially supported by the National Science Foundation under grant number ECS-8018954.

ABSTRACT

We show that for the class of graphs referred to as series-parallel at most one hamiltonian cycle is present. A linear time algorithm is proposed for producing such a cycle or alternately, concluding that the input graph is not hamiltonian. In fact, we decide the hamiltonicity issue on series-parallel graphs and in so doing, provide a proper characterization of when such graphs are hamiltonian.

1. PRELIMINARIES

In this note we consider the problem of deciding hamiltonicity on a class of biconnected graphs which are referred to as series-parallel. One source of interest stems from the variety of problems which, although difficult in general, are resolvable on graphs in the stated class. Illustrations can be found in Valdes, et. al. (1982), Takamizawa, et. al. (1982), and Rardin and Parker (1982).

Formally, a biconnected graph without loops is series-parallel if and only if it can be reduced to an edge by the sequential application of the following elementary operations:

(i) Series reduction: Replace any degree-2 vertex, k, and the incident edges (or pseudo-edges) e and f connecting k to vertices i and $j \neq i$, by a pseudo-edge, g, incident to i and j.

(ii) Parallel reduction: Replace two edges (either or both of which may be pseudo-edges) e and f, both incident to vertices i and j, by a new pseudo-edge, g, incident to i and j.

Alternately, Duffin (1965) has given the following characterization of series-parallel graphs which shall prove useful in this work.

Theorem 1: A biconnected graph G is series-parallel if and only if it possesses no subgraph homeomorphic from K_4.

■

Clearly, the conforming class of graphs is a proper subset of planar graphs since K_4 itself is planar. It is also worth observing that biconnected graphs free of K_4 homeomorphs are reducible to an edge by the series and parallel operations. It is also true that relaxation of the biconnectivity assumption produces graphs which are series-parallel per Theorem 1, yet cannot be reduced to an edge. This is of no real consequence here, however, since biconnectedness is a trivial necessary condition for hamiltonicity.

Also supporting subsequent developments is the next result which we state and prove as a lemma. Letting δ denote the minimum vertex degree in a graph, we have:

Lemma 2. If $G\{V,E\}$ is a biconnected graph without loops or multiple edges and with $\delta(G) \geq 3$, then G possesses a subgraph homeomorphic from K_4.

Proof: Let $\hat{G}$ be a subgraph in G satisfying the following property: $\hat{G}$ consists of a cycle, C^* and a path connecting a pair of vertices in C^* and passing through a nonempty set of vertices σ where $\sigma \cap C^* = \phi$. Let this path be $P_{k\ell} = (k,\sigma,\ell)$ where $k,\ell \in C^*$. Such a subgraph must exist in any graph satisfying the theorem. Let us denote by C_1 and C_2, the two subcycles in $\hat{G}$ sharing $P_{k\ell}$. Initially, let $Q \triangleq \{i: i \in C^* \backslash P_{k\ell}\}$ and select a vertex $x \in \sigma$. Since the degree of x is 2 in $\hat{G}$, there must exist an edge (x,j) in $E \backslash E(\hat{G})$. If $j \in Q$ we are done, having formed the desired homeomorph. So assume $j \notin Q$. Since G is biconnected, $G - x$ is connected, so there must be a path from j, and hence from x to some $y \in Q$, say P_{xy}. For $P_{xy} = (x,\bar{\sigma},y)$, if $\bar{\sigma} \cap V(\hat{G}) = \phi$ or if t, the first vertex of intersection with $\hat{G}$, is in Q, we are also finished, having produced K_4 homeomorph. If, however, $\bar{\sigma} \cap V(\hat{G}) \neq \phi$ and the first point of intersection

is not in Q, then this vertex, t, must lie on the path $P_{k\ell}$ (see Figure 1). However, in this case we can reduce the problem to one defined on a new graph given as either C_1 or C_2 appended with P_{xy}. New subcycles C_1 and C_2 are defined accordingly, vertices x and t are relabeled as k and ℓ, the new graph is $\hat{G}$ and the process is repeated with augmented set Q (see Figure 2). Since Q's cardinality increases by at least one each time a reduced problem is created, we must reach a point where, for a specific choice of x, the path from x to some $y \in Q$ first intersects $\hat{G}$ at a vertex $t \in Q$. In such a case we have formed a K_4 homeomorph and we are finished.

∎

There will naturally be some discretion in applying the series and parallel operations in a typical graph. Regardless, the next theorem establishes that this application can be arbitrary, i.e. the series and parallel operations are well-defined.

Theorem 3, Let $G(V,E)$ be a loopless biconnected, series-parallel graph. Then any suitable sequence of operations (i) and (ii) will reduce G to an edge.

Proof: Suppose we have for a graph satisfying the theorem, a sequence of reduction operations given as $(r_1,r_2,\ldots,r_k)$ where the sequence stops after the kth operation. Let us assume that the graph produced at this point G' is not an edge. Then since no further reductions are possible, G' contains no degree-2 vertices; that is, $\delta(G') > 3$. However, from Lemma 2, this means that G' and thus G, possesses a subgraph homeomorphic from K_4 and we contradict the assumption that G is series-parallel. Hence, for a given G, any suitable sequence of reductions will produce an edge exactly when G is series-parallel.

∎

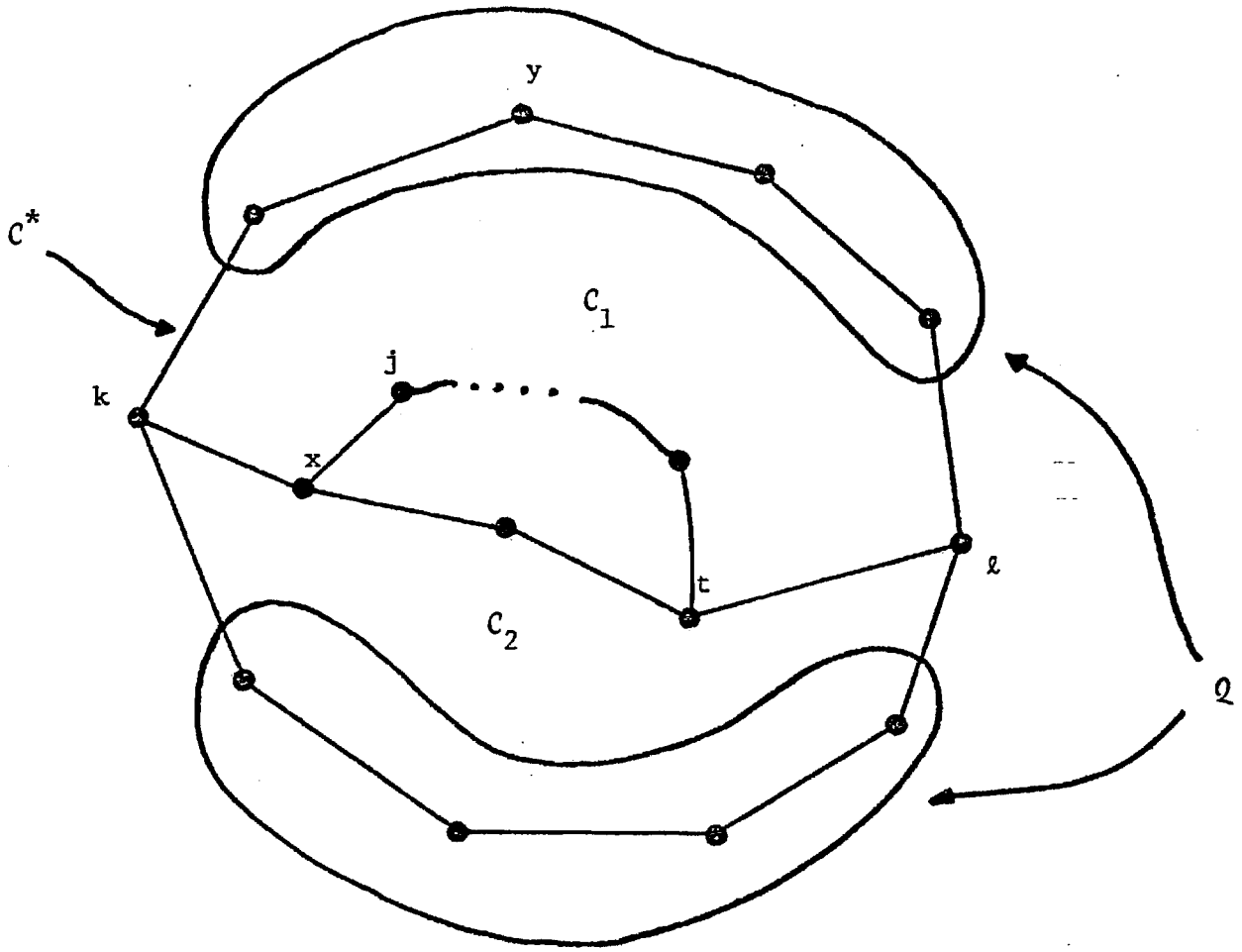


Figure 1.

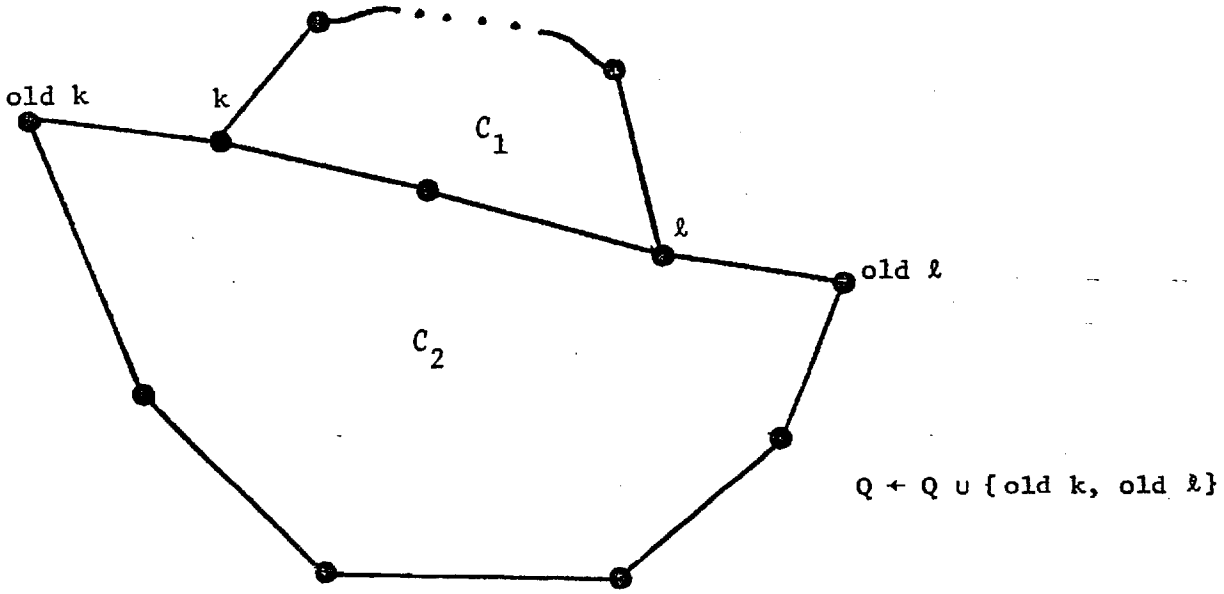


Figure 2.

2. MAIN RESULTS

Our principal result can be summarized by the following theorem:

Theorem 4. If a graph is series-parallel, then it has at most one hamiltonian cycle.

Proof. Let $G(V,E)$ be any graph having two or more hamiltonian cycles and denote two of these cycles by the vertex sequences $T = (i_1,i_2,\ldots,i_p)$ and $T' = (i_1',i_2',\ldots,i_p')$, where we can assumed without loss of generality that $i_1 = i_1'$. Let the edges implied by T and T' be $E(T)$ and $E(T')$ respectively. Now, denote by θ, the subsequence of vertices which T and T' have in common, beginning with i_1, i.e., $\theta = \{i_1,i_2,\ldots,i_t\}$, $t < p-1$. For $i_{t+1} \neq i_{t+1}'$, let $u = i_{t+1}' \in V\backslash\theta$ and create sets Q_1 and Q_2 where

$$Q_1 = \{i_k: \quad t+1 \leq k < \ell, \; i_\ell = u\}$$

$$Q_2 = \{i_k: \quad \ell+1 \leq k \leq p\}$$

Since T' is a hamiltonian cycle, it must contain at least two edges which are incident to one vertex in Q_1 and one vertex in $V\backslash Q_1$. At most one of these edges is incident to u because edge (i_t,u) is in $E(T')$. So, let (v,w) be an edge in $E(T')$ such that $w \neq u$, $w \in V\backslash Q_1$ and $v \in Q_1$. Also, we have that $w \neq i_t$ since (i_{t-1},i_t) and (i_t,u) are already in the cycle given by T'.

Hence, i_t, u, v and w are distinct vertices and (i_t,u) and (v,w) are in $E(T')$. Thus, $E \supset E(T) \cup \{(i_t,u), (v,w)\}$. However, these edges form a subgraph homeomorphic from K_4 (see Figure 3) and the other edges of E cannot destroy this property. Therefore, G is not series parallel, which establishes that no series-parallel graph can possess more than one hamiltonian cycle and the proof is complete.

■

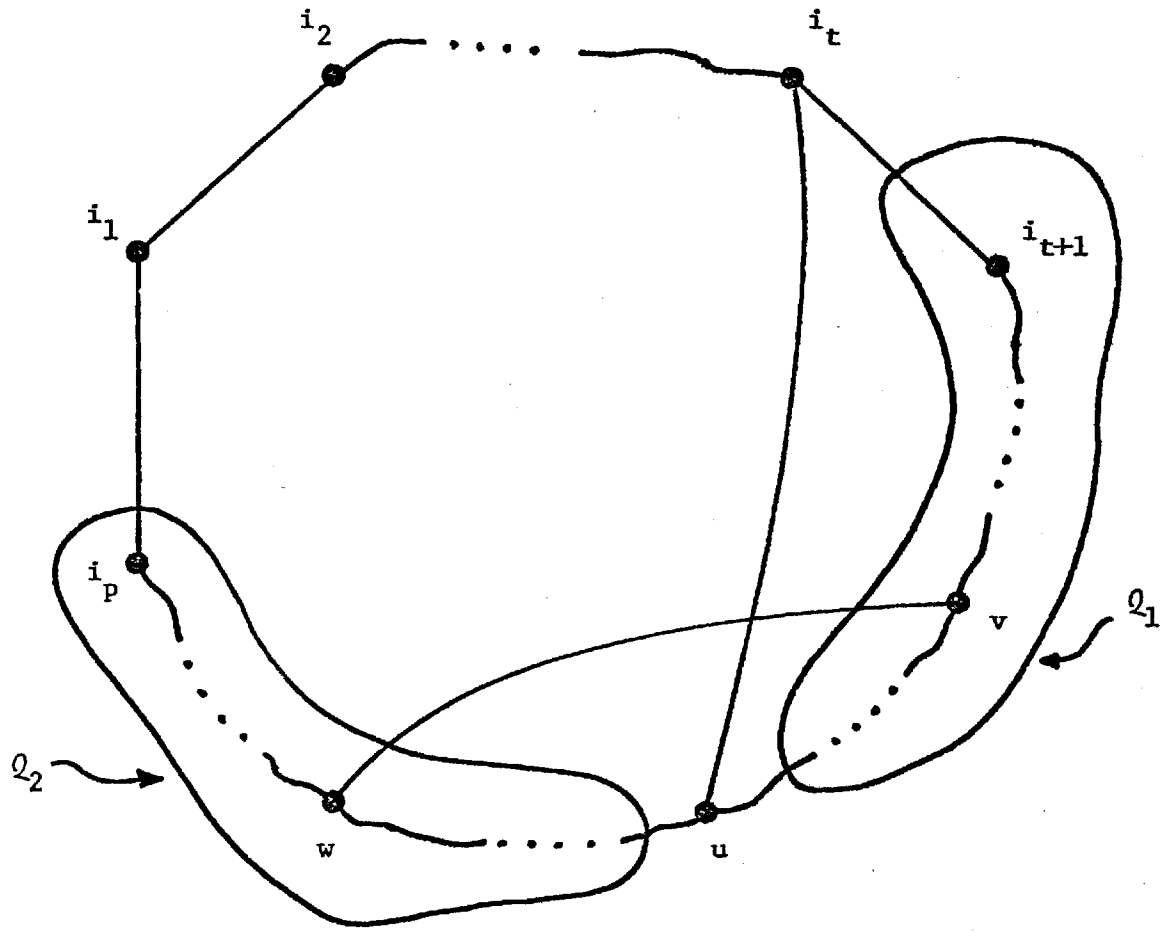


Figure 3.

The result in Theorem 4 trivially carries over for directed graphs. Here, we will call such a graph series-parallel if its underlying, undirected graph is series-parallel. Obviously, this underlying graph must be hamiltonian if the directed counterpart, $G(V,A)$, is to be as well. In such a case, we need only check the orientation around the cycle relative to A.

We have, then, that for a series-parallel graph $G(V,E)$, which is hamiltonian, the corresponding cycle is unique. Following, we state an algorithm which produces such a cycle if it exists, or concludes that the graph is not hamiltonian. In essence, the procedure decides hamiltonicity for series-parallel graphs.

Algorithm SPHAM

Step 0: Initialization. Let $G(V,E)$ be a series-parallel graph and label each edge, $e \in E$ by $\ell(e) = \{e\}$.

Step 1: Series Reduction. Locate (if possible) a degree-2 vertex in G, say k, and denote the edges (one or both of which may be pseudo) incident to k by $e_1 \triangleq (i,k)$ and $e_2 \triangleq (k,j)$. Replace e_1 and e_2 having labels $\ell(e_1) = \zeta_1$ and $\ell(e_2) = \zeta_2$ respectively, by a pseudo-edge having label $\zeta_1 \cup \zeta_2$. Call the new graph G.

Step 2: Parallel Reduction. Locate (if possible) a pair of parallel edges in G, say e_1 and e_2, and let the incident vertices be i and j. If e_1 and e_2 are both pseudo (i.e., $\min(|\ell(e_1)|, |\ell(e_2)|) \geq 2$) and G is not of order two, stop; the original graph is not hamiltonian. If one or both of the edges are pseudo and these are the only edges in G, go to (3).

Finally, if one edge is pseudo and the other is not, keep the pseudo-edge and its label, discard the other edge and let the new graph be G. Repeat this step until no parallel edges remain then return to (1).

Step 3: Stopping. The original graph has been reduced to a cycle on two vertices where either one edge is pseudo and the other is real or both are pseudo. In either case, a hamiltonian cycle in the orginal graph is obtained from the labels of the final two edges.

∎

Note that the issue regarding the actual construction of a hamiltonian cycle is left open. Clearly, ordering can be preserved and updated during the course of the algorithm or it can be accomplished at termination of the reduction operations. The efficiency of the procedure is unaffected in either case.

The correctness of SPHAM follows rather easily from earlier results in conjunction with the lemma below:

Lemma 5: Any graph H, which is homeomorphic from $K_{2,3}$ cannot be hamiltonian.

Proof. Since $K_{2,3}$ is bipartite, it cannot have an odd cycle, which certainly precludes it from being hamiltonian. Likewise, any graph homeomorphic from $K_{2,3}$ cannot be hamiltonian either, since arbitrary vertex insertions cannot possibly alter this condition.

∎

Thus we have:

Theorem 6. Algorithm SPHAM will correctly produce a hamiltonian cycle in a biconnected, series-parallel graph, G, or will conclude that none exists.

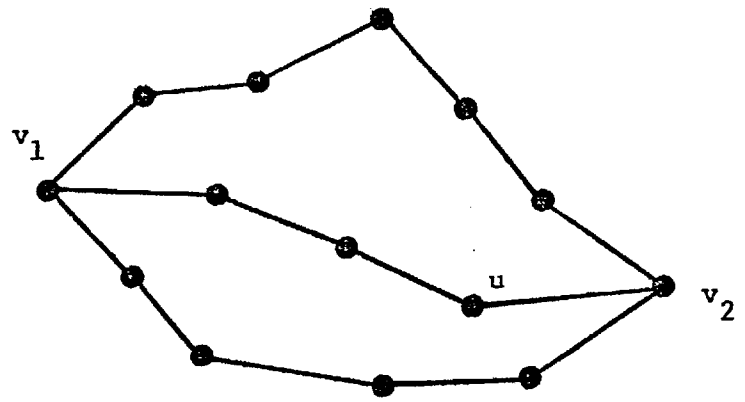
Proof: If the algorithm reaches Step 3, we clearly have decided that G is hamiltonian. Since each label represents a hamiltonian path on the subgraph corresponding to its pseudo-edge, and since all vertices of G must be in one of the two subgraphs, the desired hamiltonian cycle is easily found from the final labels.

Otherwise, the only way the algorithm can stop is in Step 2. Here, a reduced graph of order greater than two, results with two pseudo-edges in parallel. But this means that the original graph possesses a subgraph homoeomorphic from $K_{2,3}$. Let this subgraph be H (see Figure 4a) and assume without loss of generality that G itself is not homeomorphic from H since lemma 5 would preclude G from being hamiltonian. Rather, assume G to be hamiltonian and denote the vertices lying on the path from v_1 to u to v_2 in H by V_1. Now, for G to be hamiltonian, there must exist at least one path from some u to a vertex $v \in V(H) \setminus V_1$ (there may, of course, be other paths as well). Let this path be given by edge set E_{uv} (see Figure 4b). It is clear that the graph H appended by edges in E_{uv} forms a subgraph homeomorphic from K_4, which denies that G is series-parallel. Hence, no series-parallel graph possessing a subgraph homeomorphic from $K_{2,3}$ can be hamiltonian. This completes the proof.

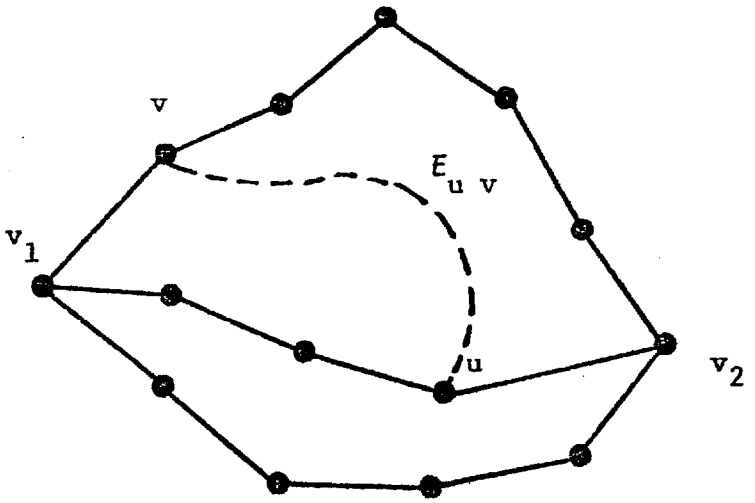
■

Algorithm SPHAM can be applied in such a way so as to require effort bounded by a function which is linear in the number of vertices and edges of the input graph. This follows from developments in Rardin and Parker (1982).

We conclude the current section with the following useful characterization.



(a)



(b)

Figure 4.

Theorem 7. A biconnected, series-parallel graph $G(V,E)$ not isomorphic to $K_4 - e$, is hamiltonian if and only if G has no subgraph homeomorphic from $K_{2,3}$.

Proof. We exclude the graph $K_4 - e$ since it is obviously hamiltonian and yet is homeomorphic from $K_{2,3}$. So in the ensuing proof, any reference to biconnected series-parallel graphs is understood to exclude $K_4 - e$.

First, suppose G is biconnected and series-parallel and possesses a subgraph homeomorphic from $K_{2,3}$. Then if G is hamiltonian, we observed from the proof of Theorem 6 that G would necessarily have a subgraph homeomorphic from K_4, contradicting the assumption that it is series-parallel.

Conversely, assume that G is not hamiltonian. Then from SPHAM, we have that termination must occur with a reduced graph on three or more vertices with two parallel pseudo-edges. Again, we saw earlier that this would mean G has a subgraph homeomorphic from $K_{2,3}$.

We have, then, that G is hamiltonian precisely when it has no $K_{2,3}$ homeomorph as a subgraph and is not hamiltonian when such a subgraph is present. This establishes the characterization and the proof is complete. ■

3. SUMMARY

We have shown that deciding hamiltonicity on series-parallel graphs is an easily resolvable issue. Further, if such a graph is hamiltonian, we know that its cycle is unique. This, in turn, implies that solving a traveling salesman problem on a series-parallel graph is indistinguishable from determining whether or not the graph is hamiltonian. Interestingly, this latter property may hold merit in the context of generating hard test problems for general-purpose traveling salesman algorithms. Some work has

been done in this area and additional insight might be available from results we have presented here.

4. REFERENCES

1. Duffin, R. J., "Topology of Series-Parallel Networks", J. Math. Anal. Appl., Vol. 10, pp. 303-318, (1965).

2. Rardin, R. L. and R. G. Parker, "Definitions, Properties, and Algorithms for Detecting Series-Parallel Graphs", ISyE Report Series, J-83, Georgia Institute of Technology, Atlanta, Ga.

3. Takamizawa, K., T. Nishizaki, and N. Saito, "Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs", J. Assoc. Comp. Mach., Vol. 29, No. 3, pp. 623-641, (1982).

4. Valdes, J., R. E. Tarjan, and E. L. Lawler, "The Recognition of Series-Parallel Digraphs", SIAM J. Computing, 11, pp. 298-313, (1982).

GUARANTEED PERFORMANCE HEURTISTICS FOR THE BOTTLENECK TRAVELING SALESMAN PROBLEM

by

R. Gary Parker
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

and

Ronald L. Rardin
School of Industrial Engineering
Purdue University
W. Lafayette, Indiana 47907

ABSTRACT

We consider constant-performance, polynomial-time, nonexact algorithms for the minimax or bottleneck version of the Traveling Salesman Problem. It is first shown that no such algorithm can exist for problems with arbitrary costs unless $P=NP$. However, when costs are positive and satisfy the triangle inequality, we use results pertaining to the squares of biconnected graphs to produce a polynomial-time algorithm with worst-case bound 2 and show further that, unless $P=NP$, no polynomial alternative can improve on this value.

Key words: graphs, combinatorics, traveling salesman, heuristic

The work reported was partially supported by the National Science Foundation under grant ECS-8018954

1. INTRODUCTION

Let $G(V,E)$ be a complete undirected graph of order $|V| \geq 3$ with weights c_{ij} on every edge (i,j) in E. Traveling Salesman Problems are defined over hamiltonian cycles in G (i.e. simple cycles including all vertices). The classic minisum version of the problem is

$$\min\left\{ \sum_{(i,j)\in H} c_{ij} : H \text{ is the edge set of a hamiltonian cycle of } G \right\}$$

Its cousin, the minimax or Bottleneck Traveling Salesman Problem (BTSP) is

$$\min \left\{ \max_{(i,j)\in H} c_{ij} : H \text{ is the edge set of a hamiltonian cycle of } G \right\}$$

It is easy to see that a polynomial-time algorithm for (BTSP) would provide a polynomial-time mechanism for testing whether arbitrary graphs are hamiltonian. Since the latter is a classic and formally difficult problem, exact polynomial-time algorithms for (BTSP) cannot exist unless $P=NP$.

It is natural, then, to seek polynomial-time, nonexact algorithms with constant performance bounds, i.e. worst-case bounds independent of problem parameters. In spite of the wide literature of such algorithms for the minisum Traveling Salesman Problem (see for example Parker and Rardin (1983a)), and the treatment of heuristic algorithms for (BTSP) in Garfinkel and Gilbert (1978), we know of no previous constant-performance-bound, polynomial-time heuristic for (BTSP).

In this note we investigate such algorithms. Our main result is a procedure with worst-case bound 2 holding when costs are positive and satisfy the triangle inequality. We also show that it is not likely that this bound will be reduced by any alternative, polynomial algorithm.

2. ARBITRARY COSTS

Sahni and Gonzales (1976) demonstrated that, unless $P=NP$, the minisum Traveling Salesman Problem admits no constant-performance-bound, polynomial-time algorithm when costs are arbitrary. A corresponding result holds for (BTSP).

Theorem 1: There can exist no polynomial-time, constant-performance-bound algorithm for an arbitrary instance of (BTSP), unless $P=NP$.

Proof: We proceed by showing that if the indicated algorithm, A, with finite bound ρ did exist, it could be employed to test hamiltonicity in arbitrary graphs— proving $P=NP$. Assume $\Omega_A/\Omega^* \leq \rho < +\infty$ where Ω_A is the value produced by algorithm A and Ω^* is an optimal value. Now, for an arbitrary graph $G(V,E)$, we can construct a corresponding instance of (BTSP) by completing the graph and assigning weights

$$c_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E \\ \rho+1 \text{ if } (i,j) \notin E \end{cases}$$

Suppose G is hamiltonian. Then in the corresponding instance of (BTSP) we have $\Omega^* = 1$ and hence $\Omega_A \leq \rho$. Conversely, if G is not hamiltonian, then $\Omega^* = \rho+1$ which implies that $\Omega_A > \rho$. Thus G is hamiltonian precisely when Ω_A is not greater than ρ, and algorithm A provides a polynomial-time

procedure for deciding which graphs are hamiltonian.

■

3. AN ALGORITHM

The negative result of Theorem 1 makes very unlikely a polynomial time, constant-performance-bound algorithm for arbitrary instances of (BTSP). However, we can derive one under more restricted costs.

3.1 Biconnected Subgraphs

A graph is said to be biconnected if every pair of its vertices belong to at least one common cycle. For a given biconnected graph $G(V,E)$ we can define the Bottleneck Biconnected Subgraph problem (BBS) as

$$\min \left\{ \max_{(i,j)\ \epsilon\ S} c_{i,j} : G(V,S) \text{ is biconnected}, S \subset E \right\}$$

It is easy to see that (BBS) provides a lower bound on (BTSP).

Lemma 1: For Ω^* = the optimal value of (BTSP) and Ω_{BB} optimal in (BBS), $\Omega_{BB} \leq \Omega^*$

Proof: Immediate from the fact that every hamiltonian cycle of a $G(V,E)$ is a biconnected subgraph.

■

Problem (BBS) is also very easily solved. A straight-forward greedy procedure gives a polynomial-time algorithm:

Algorithm BB(weighted biconnected graph $G(V,E)$)

Step 0: Initialization. Sort edges of E into nondecreasing order by edge weight c_{ij} and initialize solution set $E_{BB} \leftarrow \phi$.

Step 1: Augmentation. Select the next edge in order of the sorted list and place it in E_{BB}.

Step 2: Stopping. Test whether $G(V, E_{BB})$ is biconnected. If so, compute

$$\Omega_{BB} \leftarrow \max \{c_{ij}: (i,j) \in E_{BB}\}$$

and stop. Otherwise, repeat Step 1.

■

Lemma 2: Algorithm BB correctly computes an E_{BB} optimal in (BBS) in time bounded by a polynomial in $|E|$.

Proof: The E_{BB} solution obtained from Algorithm BB is obviously optimal because $G(V,E_{BB})$ is biconnected and construction shows every subgraph with lesser bottleneck cost is not. For polynomiality, note that Step 0 is a sort requiring $(|E| \log |E|)$ time. Steps 1 and 2 are executed on at most $|E|$ occasions, and the required check of biconnectedness at Step 2 can be done in $O(|E|)$ time (see e.g. Aho, Hopcroft and Ullman(1976)). Thus, the algorithm completes in at most $O(|E|^2)$ time.

■

3.2 Hamiltonian Cycles in the Squares of Graphs

For an arbitrary graph $G(V,E)$ the Square $G^2(V,E^2)$ is the graph formed by adding "short cut" edges for every two edge path. That is, $G^2(V,E^2)$ has the same vertex set as G, and edge set

$$E^2 \triangleq E \cup \left\{(i,k):\ (i,j,k) \text{ is a path of } G(V,E) \text{ for some } j \in V\right\}$$

The two graphs in Figure 1 illustrate the concept.

Neither the first graph in Figure 1 nor its square are hamiltonian. In fact, the tree shown establishes that connectivity in a graph is not enough to guarantee hamiltonicity of its square. If we require G to be biconnected however, the matter is different.

Lemma 3 (Fleishner (1974b)): The square of any biconnected graph is hamiltonian.

■

The fact that Lemma 3 holds was conjectured by Nash-Williams and later proved by Fleischner. Fleischner's proof is an existence one, but it yields algorithmic insights. In Rardin and Parker (1983b), we show explicitly how an algorithm can be devised from those insights to exhibit a hamiltonian circuit in the square of any biconnected graph.

Details of the procedure are far to bulky to include here. However, the approach is to derive from the given biconnected graph a particular connected and spanning subgraph possessing structural properties sufficient to make easy the construction of a hamiltonian cycle in its square. These subgraphs are defined by the edge-disjoint union of an Euler subgraph and a forest of vertex-disjoint paths. Fleishner (1974a) proved that any biconnected, bridgeless graph possesses such a subgraph and outlined how to identify a hamiltonian cycle in its (and thus the original graph's) square when, in addition, every edge meets at least one degree-2 vertex. The companion paper (1974b) inductively treats a large

number of cases in demonstrating that subgraphs with the needed degree-2 property can be obtained via suitable contraction.

Discussion in Rardin and Parker (1983b) shows that at each step of these constructions, the cardinality of at least one specified edge on vertex subset is reduced. Since steps themselves involve only polynomial exercises such as identifying the biconnected blocks of a graph, finding shortest paths and exhibiting Euler traversals of given Euler subgraphs, polynomiality of the entire algorithm is guaranteed. We summarize:

Lemma 4: Given any biconnected graph $G(V,E)$, a hamiltonian cycle $H \subset E^2$ can be produced in the square $G^2(V,E^2)$ of G in time bounded by a polynomial in $|V|$ and $|E|$.

■

3.3 The Algorithm

We are now ready to specify our nonexact algorithm for (BTSP).

Algorithm BT (Weighted Complete graph $G(V,E)$)

Step 1: Bottleneck-optimal Biconnected Subgraph. Apply Algorithm BB above to obtain $G(V,E_{BB})$, a bottleneck-optimal biconnected subgraph of $G(V,E)$.

Step 2: Tour. Identify an approximate optimal tour for (BTSP) by tracing a hamiltonian cycle, H_{BT}, in the square $G^2(V,E^2_{BB})$ of the result from Step 1, and define

$$\Omega_{BT} \triangleq \max \{c_{ij}: (i,j) \in H_{BT}\}$$

■

The algorithm certainly produces a feasible solution to (BTSP). Moreover, its polynomiality follows from Lemmas 2 and 4.

4. PERFORMANCE BOUNDS UNDER THE TRIANGLE INEQUALITY

Costs satisfy the triangle inequaltiy if $c_{ij} + c_{jk} \geq c_{ik}$ for all $i,j,k \in V$. Results of the previous section allow us to establish a constant worst-case bound on the performance of Algorithm BT in the presence of the triangle inequality.

Theorem 2: Let $G(V,E)$ be a complete undirected graph with positive weights c_{ij} satisfying the triangle inequality. Then, if Ω^* is the optimal value of (BTSP) on G, and Ω_{BT} the value produced by applying Algorithm BT to G,

$$\Omega_{BT} / \Omega^* \leq 2$$

Proof: By Lemma 1, Ω_{BB}, the value of the bottleneck-optimal biconnected subgraph produced at Step 1 of Algorithm BT satisfies $\Omega_{BB} \leq \Omega^*$ or $2\Omega_{BB}/\Omega^* \leq 2$. But edges of H_{BT}, the hamiltonian cycle obtained from Algorithm BT, either belong to E_{BB}, the optimal edge set from Algorithm BB, or correspond to two-edge paths of E_{BB}. Under the triangle inequality no edge of H_{BT} can thus cost more than $2\Omega_{BB}$. That is, $\Omega_{BT} \leq 2\Omega_{BB}$ and the proof is complete.

■

One needs only to assign weights 1 and 2 suitably to show the bound of Theorem 2 is realizable. Naturally, of course, we would prefer a smaller value than 2. Our last result shows none is likely.

Theorem 3: Let A be any polynomial-time algorithm yielding nonexact solutions for (BTSP) and Ω_A the value of solutions produced by A. If there exists a constant ρ such that $\Omega_A / \Omega^* \leq \rho$ for all (BTSP) instances satisfying the hypothesis of Theorem 2, then, unless $P=NP$, $\rho \geq 2$.

Proof: As with Theorem 1, we show that an Algorithm A with worst-case performance bound $\rho < 2$ could be used to test hamiltonicity of arbitrary graphs--proving $P=NP$. Here we choose costs

$$c_{ij} = \begin{cases} 1 \text{ if } (i,j) \in E \\ 2 \text{ otherwise} \end{cases}$$

in completing the graph. Clearly, the indicated c_{ij} satisfy the triangle inequality. Over these costs an Algorithm A with bound $\rho < 2$ would yield $\Omega_A < 2$ precisely when the given graph is hamiltonian and $\Omega_A \geq 2$ otherwise.

■

REFERENCES

1. Aho, A., J.E. Hopcroft, and J.D. Ullman (1976), The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Co., Reading, MA.

2. Fleischner, H. (1974a), "On Spanning Subgraphs of a Connected Bridgeless Graph and Their Application to DT-Graphs", J. Comb. Theory (B), 16, 17-28.

3. Fleischner, H. (1974b), "The Square of Every 2-Connected Graph is Hamiltonian", J. Comb. Theory (B), 16, 29-24.

4. Garfinkel, R.S. and K.C. Gilbert (1978) "The Bottleneck Travelling Salesman Problem:Algorithms and Probabilistic Analysis", J. Assoc. Comp. Mach., 25, 435-448.

5. Parker, R. G. and R. L. Rardin, (1983a) "The Traveling Salesman Problem:An Update of Research," Naval Research Logistics Quarterly, 30, 69-96

6. Rardin, R. and R.G. Parker (1983b), "An Efficient Algorithm for Producing a Hamiltonian Cycle in the Square of a Biconnected Graph", Industrial and Systems Engineering Report Series, J-82, Georgia Institute of Technology.

7. Sahni, S. and T. Gonzalez (1976), "P-Complete Approximation Problems", J. Assoc. Comput. Math., 23, 555-565.

ON PRODUCING A HAMILTONIAN CYCLE IN THE SQUARE OF A BICONNECTED GRAPH: AN ALGORITHM AND ITS USE

by

R. L. Rardin
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907

and

R. Gary Parker
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332

ABSTRACT

It is known that the square of any biconnected graph is hamiltonian. The proof establishing this property is given in Fleischner (1974). Unfortunately, however, Fleischner's proof is somewhat indirect and thus not immediately amenable to algorithmic implementation. In this paper, we provide a more constructive interpretation of the Fleischner result by exhibiting an efficient algorithm for producing the existing hamiltonian cycle in the stated class of graphs. Such an algorithm is important since it can be used to obtain certain polynomial-time approximation procedures possessing finite but unimprovable performance guarantees.

This material is based in part upon work partially supported by the National Science Foundation under grant ECS-8018954 and ECS-8300533.

1. INTRODUCTION

In 1974, H. Fleischner (Fleischner (1974)) proved that the square of every biconnected graph is hamiltonian. With this result Fleischner resolved a conjecture of Nash-Williams (1966) (and independently, of L. W. Beineke and M. D. Plummer). While not completely existential, Fleischner's proof is indirect, leaving vague the issue regarding the actual construction of a hamiltonian cycle in graphs satisfying the stated conditions. In the present paper, we rectify this by giving an algorithm which efficiently produces a hamiltonian cycle in the square of any biconnected graph and in this sense, makes constructive the proof of Fleischner.

Our prime interest in exhibiting such an algorithm is somewhat pragmatic. In Parker and Rardin (1983), a result is given pertaining to the absolute performance guarantee regarding any nonexact procedure for the bottleneck traveling salesman problem. This guarantee is 2, which is shown to be unimprovable by any polynomial approximation procedure unless P and NP are equivalent. The value of 2 can be acheived by employing a scheme which first constructs a bottleneck-optimal biconnected spanning subgraph after which a hamiltonian cycle in its square is sought. We discuss this notion in a subsequent section. Regardless, from Fleischner we know that such a cycle is present but less regarding a method for producing it. This paper resolves the latter issue.

2. BASIC CONCEPTS AND DEFINITIONS

Let $G(V,E)$ be a connected graph without loops or multiple edges. We say that G is biconnected if every pair of vertices lies on a cycle. Alternatively, if G is biconnected, then it possesses no cut-vertex; that is, no vertex whose removal disconnects G. Similarly, an edge in G whose

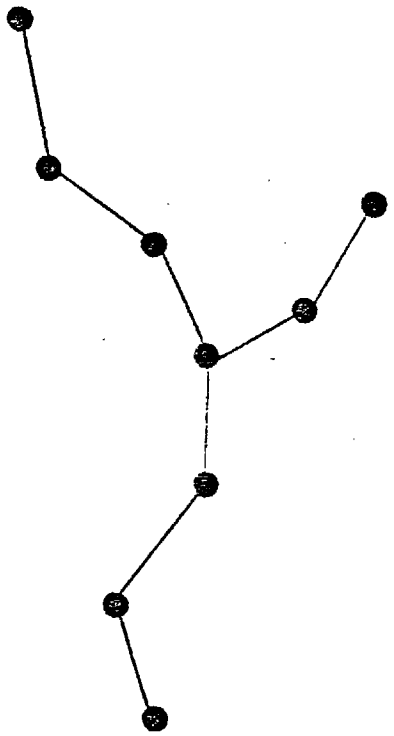
removal disconnects it is referred to as a bridge. Clearly, any connected graph having a bridge, also possesses a cut-vertex.

A connected, nontrivial graph without cut-vertices is said to be nonseparable and for a given graph, G, a maximal nonseparable subgraph is called a block. A block is edge-critical if the removal of any edge results in a subgraph which is not biconnected and finally, if every edge in G is incident to a vertex of degree two, then following Fleischner, we shall call G a DT-graph.

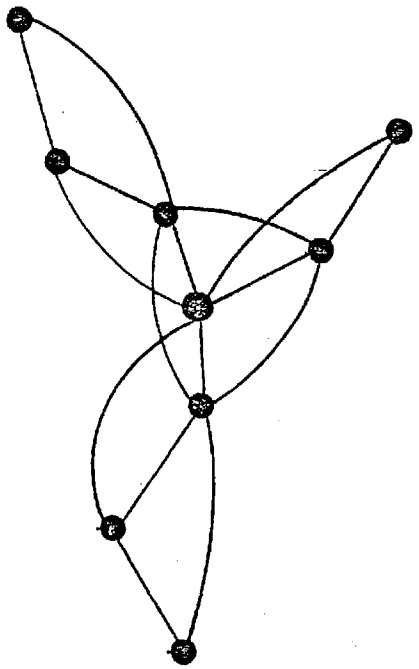
Now, consider any connected graph, G, defined on vertex and edge sets V and E respectively and let the distance between two vertices i and j in V be given as d_{ij} where d_{ij} is the length of the shortest path connecting i and j. Note that the length of a path is the number of edges in the path. Clearly, for any connected graph, d_{ij} is a metric. Now, we can define the kth power of G, given as $G^{(k)}$, to be a graph on vertex set $V^{(k)} = V$ and edge set $E^{(k)} \triangleq \{(i,j): d_{ij} \leq k \text{ in } G\}$. The graphs in Figure 1 illustrate the notion for the case k = 2. From the figure, it is clear that connectivity alone is not enough to insure that a graph's square is hamiltonian. On the other hand, it is true that the cube (k=3) of any connected graph is hamiltonian and further, a cycle in the cube can be easily constructed (e.g., Rosenstiehl (1971)).

3. THE ALGORITHM

The ensuing algorithm is fairly heavy in technical detail and quite lengthy to state. To this extent, it should be useful to begin with a concise and overly simplified statement regarding the algorithm's objective.



G



$G^{(2)}$

Figure 1. A Graph and Its Square

Essentially, Fleischner made use of the fact that every connected, bridgeless graph possesses a connected, spanning subgraph defined by the edge-disjoint union of a graph consisting of even-degree vertices only, with a forest each component of which is a path. Referred to as an EPS-subgraph, the existence of such structures in the stated class of graphs, was also established by Fleischner (1974a). Important in this regard is that a hamiltonian cycle can always be traced in the square of an EPS-subgraph and hence, in the square of the original, biconnected graph. In large measure, the bulk of the following algorithm is devoted to constructing an EPS-subgraph and the subsequent hamiltonian cycle in its square. We also note that an alternative EPS-subgraph construction is suggested in the nice paper by Lau (1981).

3.1 Main Procedure

Let the input to the algorithm be a biconnected graph $G^1(V^1, E^1)$. That we assume biconnectivity of G^1 is not limiting since checking for biconnectedness is easily accomplished (e.g., see Aho, et. al (1976)). Letting k be an index and initializing with, $k \leftarrow 1$ we can proceed.

Step 1: Case Checking. For the current graph $G^k(V^k,E^k)$, set $\mathcal{D}^k \leftarrow \{e=(x,w) \in E^k: deg^k(x) > 2 \text{ and } deg^k(w) > 2\}$ where $deg^k(i)$ denotes the degree of vertex i indicent to an edge in E^k.

1a: If G^k is a DT-graph, i.e., $\mathcal{D}^k = \phi$, go to Step 3 and begin building a hamiltonian cycle.

1b: If $\mathcal{D}^k \neq \phi$ and there is any edge $e \in \mathcal{D}^k$ such that $G^k(V^k, E^k \setminus e)$ remains biconnected, remove e from sets E^ℓ, $1 \leq \ell \leq k$ and repeat Step 1.

If neither 1a nor 1b applies, proceed to Step 2 and shrink.

Step 2: DT-Block Shrinking. Each edge $e \in \mathcal{D}^k$ is critical in that $G^k(V^k,E^k\backslash e)$ is not biconnected. For each $e \in \mathcal{D}^k$ denote by $B_1[e]$ and $B_2[e]$ the biconnected blocks of $G^k(V^k,E^k\backslash e)$ containing the defining vertices of edge e.

2a: Select as B^k the block $B^1[e]$ or $B^2[e]$ having minimum cardinality vertex set among all $e \in \mathcal{D}^k$. Denote by e^k the edge for which $B^k = B_1[e_k]$ or $B^k = B_2[e_k]$ and by w_k, the vertex of e_k belonging to B^k.

2b: Select as v_k the (unique) cut-vertex of B^k that separates it from e the remainder of $G^k(V^k,E^k\backslash e_k)$.

2c: Create graph G^{k+1} by replacing B^k in G^k with the path (w_k,a_k,b_k,v_k) where a_k and b_k are artifical vertices. Set $k \leftarrow k+1$ and return to Step 1.

Step 3: Cycle Construction. Use procedure DTHAM to construct a hamiltonian cycle, H^k, in the square of DT-graph, G^k.

Step 4: Stopping. If $k = 1$, stop; H^1 is a hamiltonian cycle in the square of G. Otherwise, go to Step 5 and restore a block.

Step 5: Block Piecing. Construct a hamiltonian cycle, H^{k-1} by first applying DTHAM to DT-graph B^{k-1} and then piecing together the result with H^k. Specific cases depend on how H^k meets vertices of the artificial path (w_k,a_k,b_k,v_k) of G^k. The appropriate treatment for each pattern is given in Table 1. After H^{k-1} is complete, set $k \leftarrow k-1$ and return to Step 4.

■

Observe that Figures 2 and 3 are useful in interpreting various cases detailed in Table 1.

Table 1: Constructing H^{k-1} from H^k and B^{k-1}

Case Number[3]	H^k Pattern on Artificial Path[1]	Required Action[1,2]
1a	...,w,a,b,v,...	Use DTHAM for a hamiltonian path and replace path (w,a,b,v) of H^k by path P(w,v).
1b	...,w,b,a,v,...	Replace path (w,b,a,v) in H^k by path (w,a,b,v). Then apply Case 1a.
2a	...,w,a,b,y,...	Use DTHAM for a hamiltonian path and replace path (w,a,b,y) of H^k by P'(w,t), (t,y).
2b	...,x,a,w,b,v,...	Replace path (x,a,w,b) in H^k by path (x,w,a,b). Then apply Case 1a.
2c	...,x,a,w,b,y,...	Replace path (x,a,w,b) in H^k by path (x,w,a,b). Then apply Case 2a.
3a	...,x,a,b,v,...,s,w,s',...	Replace path (s,w,s') by edge (s,s') and edge (x,a) by path (x,w,a) in H^k. Then apply Case 1a.
3b	...,x,a,v,b,y,...,s,w,s'...	Replace path (a,v,b,y) in H^k by path (a,b,v,y). Then apply Case 3a.

1/ Here $a \triangleq a_{k-1}$, $b \triangleq b_{k-1}$, $w \triangleq w_{k-1}$, $v \triangleq v_{k-1}$, x is the non B^{k-1} end of e_{k-1}, s and s' are neighbors of x other than w in G^k, and y and y' are neighbors of v other than b in G^k. See Figure 2.

2/ Here DTHAM produces either a hamiltonian path from w to v in the square of B^{k-1} with nonsquare edge (t,v) or a hamiltonian cycle in the square of B^{k-1} with nonsquare edges (w,z), (u,v) and (v,t). Symbols P(p,q) and P'(p,q) refer to paths in these hamiltonian entities taken counter-clockwise and clockwise around the cycle of Figure 3, respectively.

3/ Case numbering preserves that of Fleischner (1974b).

Table 1 (continued)

Case Number[3/]	H^k Pattern on Artificial Path[1/]	Required Action[1,2/]
3c	...,w,a,v,b,y,...	Replace path (a,v,b,y) in H^k by path (a,b,v,y). Then apply Case 1a.
4	...,x,a,b,y,...,s,w,s',...	Replace path (s,w,s') by edge (s,s'), and edge (x,a) by path (x,w,a) in H^k. Then apply Case 2a.
5	...,x,a,b,w,...,v,...	Use DTHAM for a hamiltonian cycle and replace path (x,a,b,w) by (x,z), P(z,u), (u,t), P(t,w).
6a	...,v,a,b,y,...,x,w,s,...	Replace path (x,w,s) by (x,s), and (a,b) by path (a,w,b) in H^k. Then apply Case 10.
6b	...,v,a,b,y,...,s,w,x,...	Replace path (s,w,x) by (s,x), and (a,b) by path (a,w,b) in H^k. Then apply Case 10.
6c	...,v,a,b,y,...,s,w,s',...	Replace path (s,w,s') by (s,s'), and (a,b) by path (a,w,b) in H^k. Then apply Case 10.
7	...,x,a,v,...,y,b,w,...	Use DTHAM for a hamiltonian cycle and replace path (x,a,v) by (x,z), P(z,v) and path (y,b,w) by (y,t), P(t,w).
8	...,x,a,v,...,w,b,y,...	Use DTHAM for a hamiltonian cycle and replace path (x,a,v) by (x,z), P(z,v) and path (w,b,y) by P'(w,t), (t,y).
9	...,x,a,v,b,w,...	Use DTHAM for a hamiltonian cycle and replace path (x,a,v,b,w) by (x,z), P(z,w).
10	...,y,b,w,a,v,...	Use DTHAM for a hamiltonian cycle and replace path (y,b,w,a,v) by (y,t), P(t,v).
11a	...,w,a,x,...,y,b,v,...	Replace path (y,b,v) by (y,v) and (w,a) by path (w,b,a) in H^k. Then apply Case 5.

Table 1 (continued)

Case Number[3]	H^k Pattern on Artificial Path[1]	Required Action[1,2]
11b	...,w,a,x,...,v,b,y,...	Replace path (v,b,y) by (v,y), and (w,a) by path (w,b,a) in H^k. Then apply Case 5.
11c	...,w,a,x,...,y,b,y',...	Replace path (y,b,y') by (y,y'), and (w,a) by path (w,b,a) in H^k. Then apply Case 5.
12	...,w,a,v,...,y,b,y',...	Replace path (y,b,y') by (y,y') and (a,v) by path (a,b,v) in H^k. Then apply Case 1a.
13	...,x,a,v,...,y,b,y',...	Replace path (y,b,y') by (y,y'), and (a,v) by path (a,b,v) in H^k. Then apply Case 3a.

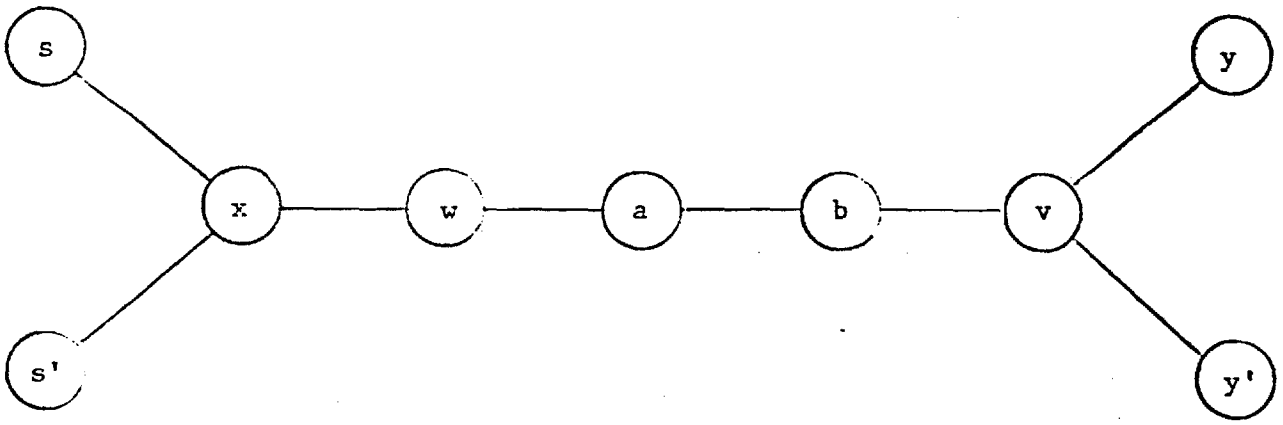


Figure 2: Vertex Arrangement Around Artificial Path

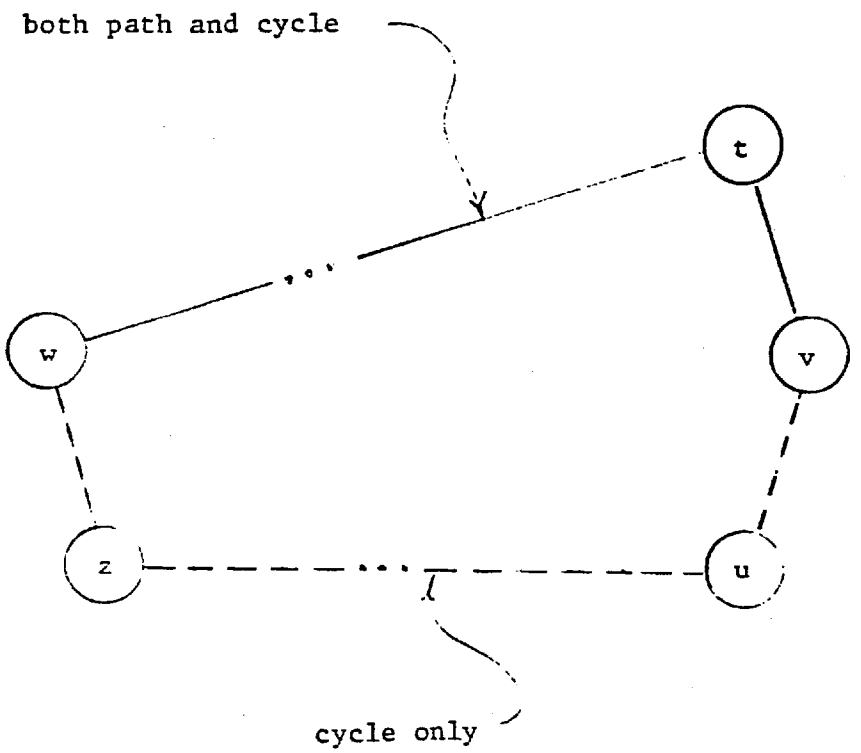


Figure 3: Vertex Sequence of Hamiltonian Paths and Circuits in the Square of B^{k-1} Produced by DTHAM

3.2 Procedure DTHAM

Clearly, Step 3 of the main procedure is crucial. In this section, we specify the relevant routine, DTHAM.

Step 0: Initialization. Let G_D be the current DT-graph and denote two distinguished vertices, say v and w which are in the same block of G_D and have only degree-2 neighbors. If a hamiltonian path is desired, add (unless it is already present) an artificial edge, (v,w), to G_D.

Step 1: EPS-Subgraph Construction. Use procedure EPS to find a spanning subgraph S of G_D such that

(i) S is the union of an Euler subgraph E and a forest of (vertex) disjoint paths P

(ii) E and P are edge-disjoint

(iii) Given vertex v belongs to E, but not to P.

(iv) Given vertex w belongs to E, and is not an internal vertex of P.

(v) If v and w are adjacent in G_D, then v and w are adjacent in E.

Step 2: Reduction. If every edge of P separates S into disjoint components, proceed to Step 3. Otherwise delete an edge that does not and repeat this step.

Step 3: Mate Edges. Subgraph S can now be viewed as consisting of a tree of components of E linked by segments of paths in P. Select a minimum cardinality set M of edges in P which breaks all such links, i.e., divides S into components each containing exactly one component of E. Then construct subgraphs $S_1,\ldots,S_n$ of S from the components $E_1,E_2,\ldots,E_n$ and $P_1,P_2,\ldots,P_n$ induced by M, duplicating edges of M so that each belongs to both its adjacent P_i.

Step 4: Cycles in Components. For each component S_i of Step 3 determine a hamiltonian cycle H_i in the square of S_i as follows:

4a. If S_i contains artificial edge (v,w) pick as t an artificial vertex inserted in edge (v,w). Otherwise t is any degree-2 vertex of the Euler component E_i in S_i.

4b. Construct an Euler tour T_i of the subgraph E_i so that T_i begins and ends at t.

4c. Beginning with $t_0 \leftarrow t$ trace T_i until t recurs, constructing H_i as indicated below ($deg(\cdot)$ refers to degree in S_i; t_0,t_1,t_2 are the present and next two vertices of T_i)

Case on t_0,t_1,t_2	Evolution of H_i
(i) $deg(t_1) = 2$	(t_0,t_1)
(ii) $deg(t_1) > 2$, $t_0 \neq t$, and t_1 will be revisited in T_i or has already been visited in H_i	(t_0,t_2)
(iii) $deg(t_1) > 2$, $t = t_0$ or t_1 will not be revisited in T_i, and $t_1 \in E_i$ P_i	$(t_0,t_1)(t_1,t_2)$
(iv) $deg(t_1) > 2$, $t = t_0$ or t_1 will not be revisited in T_i, and t_1 is an end vertex of P_i	(see Figure 4)
(v) $deg(t_1) > 2$, $t = t_0$ or t_1 will not be revisited in T_i, and t_1 is an internal vertex of P_i	(see Figure 5)

Step 5: Solution. By constructions of Figures 4 and 5, each H_i contains all end edges of S_i including those in M. Construct a hamiltonian cycle H in the square of S (and thus in G_D) by $H = \bigcup_{i=1}^{n} H_i \backslash M$. If only a hamiltonian path is desired, reduce H to a path by removing the artificial path (v,t,w).

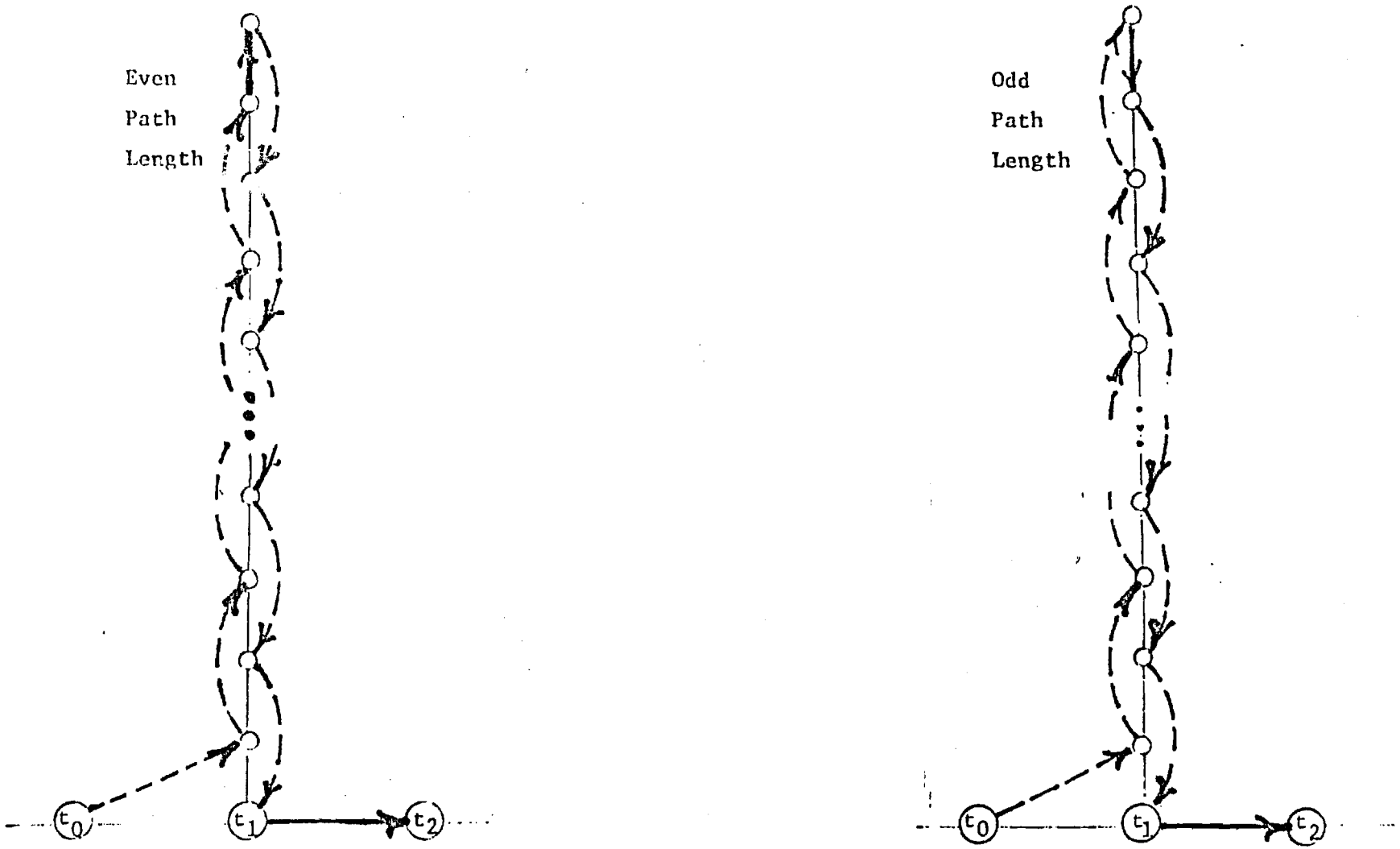


Figure 4: Pattern of Traversal
When t_1 is Path End

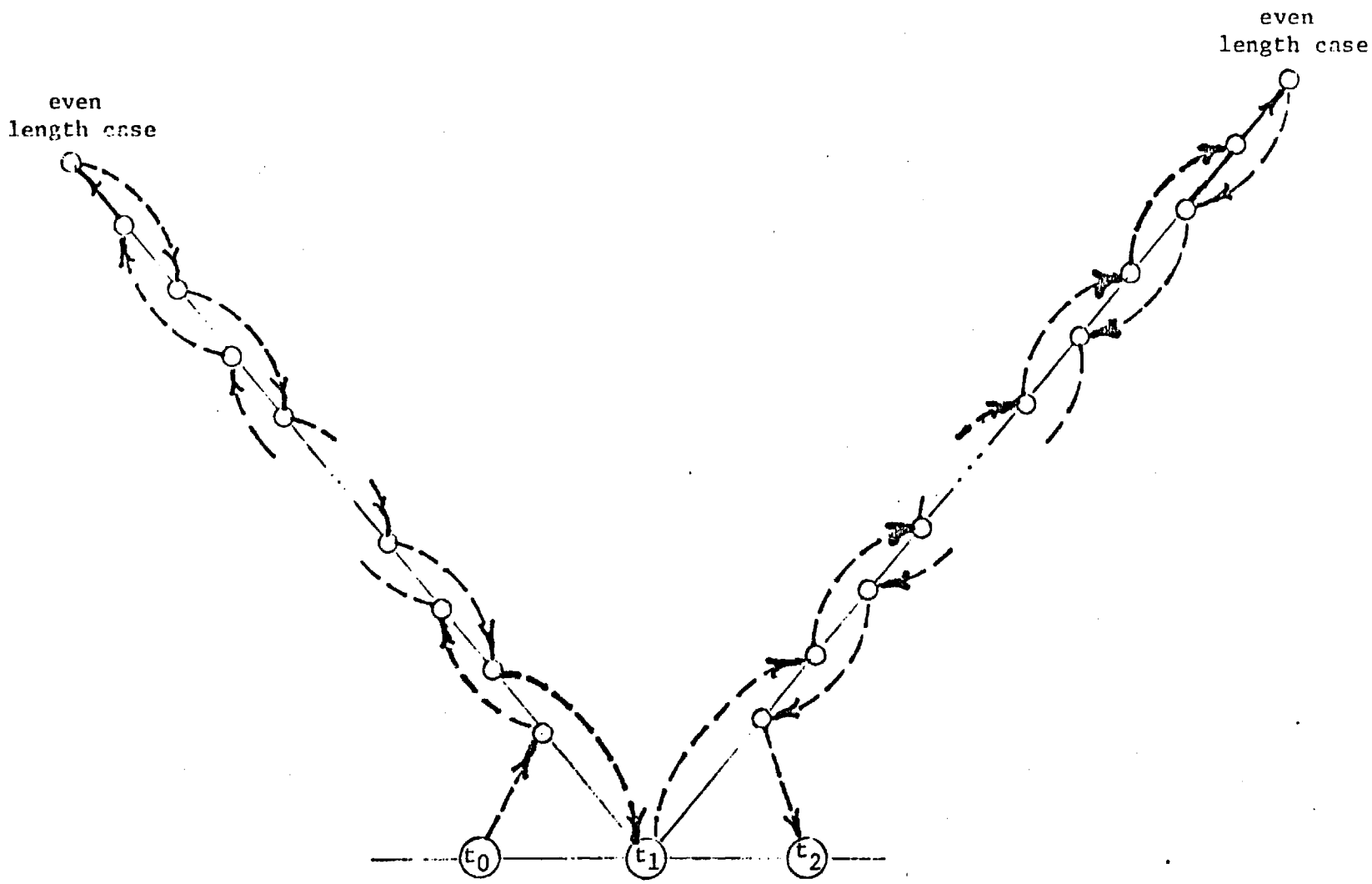


Figure 5: Pattern of Traversal When t_1 Is an Internal Path Vertex

3.3 EPS - Subgraph Construction.

Just as DTHAM acts as a subroutine to the main procedure, Step 1 of DTHAM can be treated similarly. Below, we give routine EPS.

Step 0: Initialization. Let $G = G_D$ be a biconnected graph with two distinguished vertices v and w. Operationally, v and w are the two specified vertices of G_D from Step 0 of DTHAM. Begin a list of unprocessed subgraph sets given by the 4-tuple $(G_\ell, v_\ell, w_\ell, C_\ell)$ where initially,

$G_1 \leftarrow G$

$v_1 \leftarrow v$

$w_1 \leftarrow w$

$C_1 \leftarrow$ any cycle of G containing v and w, and edge (v,w) if present in G.

Step 1: Decomposition Stopping. If the list of unprocessed subgraph sets contains only G_ℓ that are cycles or single edges, go to Step 4 and begin reassembly. Otherwise, pick $(\bar{G}, \bar{v}, \bar{w}, \bar{C})$ from the list with $\bar{G}$ not an edge or a cycle.

Step 2: Preprocessing. If $\bar{G}$ contains an edge e such that $\bar{G} - e$ is biconnected, remove e from $\bar{G}$ and repeat this step.

Step 3: Decomposition. If G is not now only the cycle $\bar{C}$, process G by decomposing it into two or more new entries in the unprocessed subgraph list as follows:

3a: If $\bar{G}-\bar{C}$ is biconnected add G_0 and G_1 to the list with

$G_0 \leftarrow C_0 \leftarrow \bar{C}$

$v_0 \leftarrow \bar{v}$

$w_0 \leftarrow \bar{w}$

$G_1 \leftarrow \bar{G} - \bar{C}$

$v_1 \leftarrow$ $\bar{v}$ if $\bar{v}$ belongs to $\bar{G} - \bar{C}$, or any vertex of $\bar{G} - \bar{C}$ except $\bar{w}$ otherwise

$w_1 \leftarrow$ $\bar{w}$ if $\bar{w}$ belongs to $\bar{G} - \bar{C}$, or any vertex of $\bar{G} - \bar{C}$ except v_1 otherwise

C_1 any cycle of G_1 containing v_1 and w_1

3b: If $\bar{G} - \bar{C}$ has disconnected components $\bar{Q}_1, Q_2, \ldots, Q_q$ add $G_1, G_2, \ldots, G_q$ to the unprocessed subgraph list with

$G_i \leftarrow \bar{Q}_i \cup \bar{C}$

$v_i \leftarrow \bar{v}$

$w_i \leftarrow \bar{w}$

$C_i \leftarrow \bar{C}$

3c: If $\bar{G} - \bar{C}$ is connected but not biconnected and one end block B_1 of $\bar{G} - \bar{C}$ (i.e., one block with a single cut-vertex c_1) does not contain v or w except possibly as its cut-vertex, $((B_1 - c_1) \cap \{v,w\} = \phi)$, choose the least cardinality block chain $B_1, B_2, \ldots, B_b$ of $\bar{G} - \bar{C}$ beginning with B_1 and leaving $\bar{G} - \bigcup_{i=1}^{b} B_i$ biconnected. Let c_0 be any vertex of B_1 except c_1; c_i be the cut vertex joining B_i to B_{i+1}, i=1,2,...,b-1; and c_b be the cut vertex joining $\bigcup_{i=1}^{b} B_i$ to the remainder of $\bar{G} - \bar{C}$. Then create b or b+1 new entries in the unprocessed subgraph list by

$$G_0 \leftarrow \bar{G} - \bigcup_{i=1}^{b} B_i$$

$$v_0 \leftarrow \bar{v}$$

$$w_0 \leftarrow \bar{w}$$

$$C_0 \leftarrow \bar{C}$$

and for i=1,2,...,b or i=1,2,...,b-1 if B_b is a single edge

$$G_i \leftarrow B_i$$

$$v_i \leftarrow c_i$$

$$w_i \leftarrow c_{i-1}$$

$$C_i \leftarrow \begin{cases} \text{any cycle of } G_i \text{ containing } v_i \text{ and } w_i \text{ if } G_i \text{ is biconnected, or} \\ \phi \text{ if } G_i \text{ is a single edge} \end{cases}$$

3d: If $\bar{G} - \bar{C}$ is connected but not biconnected and its only two end blocks each contain one of $\bar{v}$ and $\bar{w}$ at other than their cut points, let $B_1, B_2, \ldots, B_b$ be the block chain forming $\bar{G} - \bar{C}$ with $\bar{w} \in B_1$ and $\bar{v} \in B_b$. Also define $c_0 \leftarrow \bar{w}$, $c_b \leftarrow \bar{v}$ and for i=1,2,...,b-1 pick c_i as the cut vertex joining blocks B_i and B_{i-1}. Then create b or b+1 new entries in the unprocessed subgraph list exactly as in Step 3c.

After processing $\bar{G}$ in one of the above ways or skipping $\bar{G}$ if it is only the cycle $\bar{C}$, return to Step 1.

Step 4: Initial EP-Subgraphs. Each subgraph G_ℓ in the unprocessed list is now either a cycle C_ℓ or an edge e_ℓ. Generate spanning EP (Euler-path) subgraphs for each as follows:

4a: If $G_\ell = C_\ell$, choose Euler subgraph $E_\ell \leftarrow C_\ell$, and path forest

subgraph $P_\ell \leftarrow \phi$.

4b: If $G_\ell = e_\ell$, choose Euler subgraph $E_\ell = \phi$, and path forest subgraph $P_\ell = e_\ell$.

Step 5: EP-Subgraph Reassembly. Taking the processed subgraphs G_ℓ in reverse order of their creation, construct a spanning EP-subgraph for each G_ℓ by taking the union of all E_t and P_t where G_t was created by decomposing G_ℓ. Specifically, $E_\ell = \bigcup_t E_t$, $P_\ell = \bigcup_t P_t$ except that any paths $p_i \in P_i$ and $p_j \in P_j$ sharing a common (end in both paths) vertex of $G_i \cap G_j$ are replaced by the single path $p_i \cup p_j$.

■

4. AN EXAMPLE

We can demonstrate the algorithm detailed in the previous three subsections by considering the biconnected graph in Figure 6. Letting this graph be denoted by $G^1(V^1,E^1)$ we proceed in step-by-step fashion.

Step 1. Initially, we have $\mathcal{D}^1 = \{(4,7), (10,16), (12,13)\}$. Removing edge (4,7) leaves a biconnected graph and thus we set $\mathcal{D}^1 \leftarrow \mathcal{D}^1 \setminus (4,7)$ and $E^1 \leftarrow E^1 \setminus (4,7)$. Relative to the new G^1, removing either edge (10,16) or (12,13) destroys biconnectedness and since $\mathcal{D}^1 \neq \phi$ we perform a shrinking operation.

Step 2. We have for each edge in $\mathcal{D}^1$, the two blocks shown in Figure 7. Let us select arbitrarily the minimum cardinality one induced by $e_1 \triangleq (12,13)$; that is, let B^1 be the cycle (13,14,16,15). Accordingly, we have $w_1=13$ and $v_1=16$. Graph G^2 appears as in Figure 8 where B^1 is replaced by the artificial path (w_1, a_1, b_1, v_1) as depicted.

Step 1. Since $\mathcal{D}^2 = \phi$, G^2 is a DT-graph and we can proceed with the construction of a hamiltonian cycle.

Step 3. We seek cycle H^2 in the square of G^2 and thus, call routine DTHAM

using G^2 as input. Accordingly, we proceed to Step 1 of DTHAM which requires the construction of an EPS-subgraph in G^2.

Step 0 (EPS). Let us denote the first unprocessed subgraph set by the 4-tuple (G_1,v_1,w_1,C_1) where $G_1 = G_2$, v_1 and w_1 are as shown in G^2, and C^1 is the cycle $(12,11,10,v_1,b_1,a_1,w_1)$.

Step 1 (EPS). Selecting the unprocessed subgraph set just constructed, we have $\bar{G} = G_1 = G^2$ and $\bar{G}$ is decomposed.

Step 3c (EPS). We identify blocks B_i, $1 \leq i \leq 6$ relative to $G^2 \setminus E(C_1)$ as shown in Figure 9. We also denote the respective vertices c_j for $j=0,1,\ldots,6$. Since B_6 is a single edge, we create new subgraph set entries $(G_0,v_0,w_0,C_0),\ldots,(G_5,v_5,w_5,C_5)$. These are shown in Figure 10 where, for ease, only the relevant subgraphs are displayed.

Step 1 (EPS). Since the new list of unprocessed sets contain only cycles or edges we can begin the reassembly process.

Step 4 (EPS). Relative to the graphs displayed in Figure 10 we can construct E_ℓ and P_ℓ for $\ell=0,1,\ldots,5$ using the rules 4a and 4b. We have:

$\ell = 0$: $E_0 = (C_0)$, $P_0 = \phi$

$\ell = 1$: $E_1 = \phi$, $P_1 = (5,12)$

$\ell = 2$: $E_2 = \phi$, $P_2 = (2,5)$

$\ell = 3$: $E_3 = (C_3)$, $P_3 = \phi$

$\ell = 4$: $E_4 = (C_4)$, $P_4 = \phi$

$\ell = 5$: $E_5 = \phi$, $P_5 = (7,9)$

Step 5 (EPS). Since the only processed subgraph was the original one (Figure 8), the desired EP-subgraph is easily reconstructed as shown in Figure 11. We now return to DTHAM.

Step 3 (DTHAM). In S, let us form M as the single edge denoted by P_2 in

Figure 11. This induces (per the stated construction of the step) subgraphs S_1 and S_2 shown in Figure 12.

Step 4 (DTHAM). We denote (arbitrarily) by t, a degree-2 vertex in S_1 and S_2 and construct eulerian cycles T_1 and T_2 accordingly. These cycles are denoted by dotted edges in Figure 12. The cycles H_1 and H_2 are generated using the stated rules and result as shown in Figure 13.

Step 5 (DTHAM). Patching together H_1 and H_2 as specified, yields hamiltonian cycle H^2 as shown in Figure 14.

Step 4 (Main). Since $k \neq 1$, we must restore a shrunken block.

Step 5(Main). Since B^1 was shrunk earlier and replaced by an artificial path, creating G^2, we observe from case 1a of Table 1 that a hamiltonian path from vertex w_1 to vertex v_1 through the square of B^1 is needed. We can find such a path by employing DTHAM; however, for ease we shall simply select the path (13,15,14,16). Here, w_1=13 and v_1=16. Finally, replacing the artificial path (w ,a ,b ,v) by the stated one, produces cycle H^1 in G^1 and the procedure is complete. We leave it to the reader to make this replacement and moreover, to verify that H^1 is a suitable hamiltonian cycle in the square of G^1.

■

5. EVALUATION OF THE ALGORITHM

In this section, we examine the veracity and computational requirements of the algorithm detailed earlier. In both, we concentrate only on the more crucial points of verification.

5.1 Validity of the Procedure

Our discussion is organized around the three-component breakdown by which the algorithm was presented previously.

Main procedure. Graph G^1 is biconnected by construction. It possesses

no multiedges because G doesn't and no bridges because it would not then be biconnected. After finitely many applications of Steps 1 and 2 a DT-graph must result. This follows since Steps 1b and 2 both reduce $\mathcal{D}^k$. In the latter case edge e_k is in $\mathcal{D}^k$ and after step 2, it is not because w_k is now degree 2.

Now, to allow the construction of Step 5, it must be true of each B^k that

(i) $w_k \neq v_k$

(ii) B^k contains only one cut vertex of $G^k(V^k, E^k \setminus e_k)$

(iii) B^k is a DT-graph

(iv) all neighbors of w_k in B^k have degree 2 there

(v) all neighbors of v_k in B^k have degree 2 there

Fleischner (1974b) establishes these properties in Theorem 1 and Remark 1. Furthermore, the cases in Table 1 are derived from ones given by Fleischner except for cases 1b, 2b, 2c, 3b, and 3c which have been added in order to enumerate ones excluded (in Fleischner (1974b)) by Figure 2.

DTHAM. The mating process of Step 5 is valid because edges of M may be viewed as edges of a tree linking components S_i. Also, Figures 3 and 4 show end edges are always part of the tour. The implied hamiltonian cycle has 2 true edges meeting v because property (iii) of Step 1 assumes v is not connected with any path. Thus on the last visit to v, case (iii) of Step 4c will apply. Similarly, the computed hamiltonian cycle will have at least one true edge meeting w. By (iv) of Step 1, w is either an identical case to v or at the "foot" of a structure like that in Figure 3.

Addition of the artificial edge in hamiltonian path cases assures v and w are both at least degree 3 without destroying the DT structure since all neighbors of v and w are degree 2. In hamiltonian path cases we start at the middle of (v,w) and proceed first to v. Thus (t,v) is in H. When we return through w, either case (iii) or case (iv) of Step 4 applies and both place (w,t) in the tour. Thus, (v,t,w) is in the tour to delete at Step 5. There is also one other nonequal edge at v.

EPS. Cycles C_k of the original subgraph and all subsequent ones always recur in the generated Euler system E_k. Thus, particular, v and w end in E_k and so does edge (v,w) if present in G.

The unioning process of Step 5 always combines $S_i \triangleq E_i \cup P_i$ and $S_j \triangleq E_j \cup P_j$ into an EP-subgraph $S_k \triangleq E_k \cup P_k$ either because P_i and P_j are vertex disjoint and $S_i \cap S_j$ is a subset of vertices or an Euler subgraph $E_i \cap e_j$, because $S_i \cap S_j$ is a single vertex not internal to a path of either P_i or

In Fleischner (1974a) Lemmas 1, 2 and 3 verify these facts. Specifically, when we restore a Step 3a decomposition, one of S_i and S_j is a cycle. Thus $P_i \cap P_j = \emptyset$ and $S_i \cap S_j$ is subset of vertices. If the decomposition was by Step 3b, the subgraphs have only cycle $\bar{C}$ in their intersection -- an Euler subgraph. This is true because the remainder of the S_i belong to disjoint Q_i. If the decomposition was at Step 3c, we first union EP-subgraphs for each block into say S_1 and then combine with S_0 of G_0. The block subgraphs have only a single cut-vertex in common, and it is always a v-vertex in one, implying it not to be path internal. Finally, $S_0 \cap S_1$ is a subset of vertices of $\bar{C}$ plus (if b+1 subgraphs were generated) the vertex c_b. We have chosen c_b as a v_b so that it cannot be an internal path vertex. Also, vertices of $\bar{C}$ common to B_1 are degree 2 in G_0 and thus

cannot have incident paths. The decomposition of Step 3d is similar to 3a and 3c. Blocks are combined as in 3c; S_0 is a cycle as in case 3a.

The entire decomposition stops because all subgraphs G_ℓ produced in processing $(\bar{G},\bar{v},\bar{w},\bar{C})$ have $|G_\ell - \bar{C}| < |\bar{G}-\bar{C}|$.

Finally, we want w and v in the "E-part" of the final subgraph, v incident to no path of p and w at most a path end. These properties follow because we always keep v and w on the cycle C_k when both are present in a G_k and $C_k \subset E_k$. Moreover, our choice of v and w as lone cycle vertices or cut vertices in the various decompositions always avoids undesired paths.

5.2 Computational Requirements

The graph produced after the (finite) application of Steps 1 and 2 contains entirely original edges or a mixture of original and artificial ones created by block shrinking. Here, for each block shrunk, a 3-edge path is created and thus the respective $\mathcal{D}^k$ is reduced. Biconnectedness checking can be efficiently performed and hence, DT-graph construction requires effort bounded by a polynomial in $|V|$ and $|E|$.

Now, for a given EPS-subgraph (of a DT-graph), the construction of a hamiltonian cycle in its square requires first a reduction and edge mating process (Steps 2 and 3 of routine DTHAM) both of which are clearly polynomial in the size of P, the path component in the EPS-subgraph. Of course, we must produce eulerian cycles in subgraphs E_i but this is easy and finally, for each subgraph S_i induced in step 3 of DTHAM, a hamiltonian cycle in its square is obtained from the eulerian cycles in the respective E_i and the rules of step 4 (DTHAM) and Figures 4 and 5. This along with the Step 5 (DTHAM) patching process is certainly polynomial in the number of edges in the EPS-subgraph.

Turning specifically to the EPS-subgraph construction (routine EPS), we see that crucial in the entire process is the decomposition step (step 3). Throughout, biconnectedness checking is performed but as before, this does not affect overall polynomiality. We need only demonstrate that the number of unprocessed subgraphs formed in step 3 is polynomial in the size of the input DT-graph. Let us take the component steps in order.

In 3a, one finished (cycle) graph is produced as well as one biconnected subgraph in which a new cycle is selected. Hence, the number of non-cycle edges will decrease by at least 3. In 3b, the set of non-cycle edges is q-sected and by the construction in this step, $(\bar{Q}_i \cup \bar{C})$-edges may be added. However, each unprocessed subgraph created in this manner must next be processed by one of the other three cases. In 3c, the set of non-cycle edges is b-sected and edges are simply transferred to new cycles or to biconnected blocks. No edges are added by the unprocessed subgraph creation. Step 3d is similar to 3c differing only in the block-chain specification. Important in this regard is that no edges are added in the unprocessed subgraph construction.

Now, in order to evaluate the overall effort of step 3 in procedure EPS, we can consider a simple progress measure (x,y) where x is the number of non-cycle edges in all unprocessed subgraph 4-tuples and y, the number of subgraphs into which non-cycle edges are subdivided. Clearly, x is of size $O(|E|)$ and within each, y can be this large as well, rendering total computation for this step at worse, $O(|E|^2)$.

Of course, the entire process must be repeated k times per steps 4 and 5 of the main procedure. This however, clearly preserves the order of the overall algorithm since the block-piecing computation requires only case

checking which is detailed in Table 1. We may thus conclude that the entire procedure can be performed in time bounded by a polynomial in the size of the input graph.

6. EMPLOYMENT OF THE ALGORITHM

We suggested at the outset that a principal interest in producing (efficiently) hamiltonian cycles in the stated class of graphs was to permit construction of approximation algorithms for various hard combinatorial optimization problems. In this section, we briefly describe one such construction which originated with the authors ([7]). We also note that the basic approach using the guarantee of hamiltonicity in biconnected squares has been employed elsewhere (e.g. Hochbaum and Shmoys (1983)).

The bottleneck traveling salesman problem (BTSP) seeks a hamiltonian cycle in a weighted graph the largest edge weight of which is minimized. The problem is known to be difficult - indeed it can be easily shown to be equivalent to the problem of deciding which graphs are hamiltonian.

The formal intractibility of the (BTSP) makes its treatment by approximation schemes particularly legitimate. Accordingly, for such nonexact procedures we would like to fix (finite) bounds on their worst-case performances. It is here that we can employ our algorithm of Section 3.

First, it is easy to show that no finite bound for any (polynomial) heuristic can exist for instances of (BTSP) without edge weights satisfying the triangle inequality unless P and NP are equivalent. Thus, we can consider our instances to be defined on complete graphs, K_n, where edge weights in fact satisfy the triangle inequality.

Now, since any hamiltonian cycle is biconnected but not the converse, let us construct an optimal bottleneck biconnected (spanning) subgraph of K_n. Thus, if α is the value of the maximum weight edge in this subgraph then the bottleneck optimal value of the corresponding instance of the BTSP on K_n can be no less than α. Most important here is that the stated bottleneck optimal biconnected subgraph is easy to obtain. We simply apply a greedy procedure to the list of edges in K_n arranged in nondecreasing order of edge weights. Beginning with the empty graph on n vertices, edges are added in order with termination occurring when the first spanning, biconected subgraph is constructed. Clearly, such a scheme is optimal and its efficiency follows since checking for biconnectivity is easy.

Letting $\hat{G}$ be our bottleneck optimal subgraph, then $\hat{G}$ is suitable input to the algorithm of Section 3. That is, $\hat{G}$'s biconnectivity guarantees hamiltonicity of its square and such a cycle, $\hat{H}$, will be efficiently produced by the algorithm. Moreover, $\hat{H}$ is an approximate solution to the given instance of the BTSP. If we let v(BTSP) be the optimal BTSP value and $v(\hat{H})$ the value produced by the heuristic, then $v(\hat{H}) \leq 2\alpha$ which follows from the triangle inequality and thus $v(\hat{H}) \leq 2v(BTSP)$.

So, as claimed in the introduction, we can produce a solution to any instance of the stated BTSP in polynomial time which differs from an optimal solution by a factor at most 2. We also claimed that this bound was unimprovable by any polynomial alternative unless $P = NP$. That this must be so, follows from the obvious use of any alternative BTSP heuristic for deciding hamiltonicity in arbitrary graphs. We simply create a corresponding BTSP instance, weighting edges by 1 or 2 depending on whether or not an edge is present in an instance upon which hamiltonicity is to be tested. Such edge weights clearly satisfy the triangle inequality and we

would apply the hypothesized BTSP heuristic. If the graph in question is hamiltonian then the optimal bottleneck value would be 1 and the assured heuristic would produce it (recall, such a procedure is assumed to have worst-case bound strictly less than 2). Alternately, if the graph is not hamiltonian, then the corresponding optimal BTSP value would be 2 which must again be the value produced by the heuristic (edges have weights confined to 1 or 2). Thus, we need ony observe the value produced by the heuristic and the hamiltonicity of the original graph is decided accordingly. This problem is *NP*-complete however, and the existence of such a BTSP heuristic would render *P* and *NP* equivalent.

7. SUMMARY

In this paper, we have addressed ourselves primarily to the problem of producing hamiltonian cycles in the squares of biconnected graphs. Existence of such cycles was resolved earlier by Fleiscshner, but their explicit construction was less obvious. The computational procedure given here rectifies this.

We also have demonstrated (without detail) how the stated algorithm - indeed, the Fleischner result itself, can be used in the development of nonexact or approximation procedures. In this regard, it would appear that further exploration is warranted, especially in the context of performance bound construction.

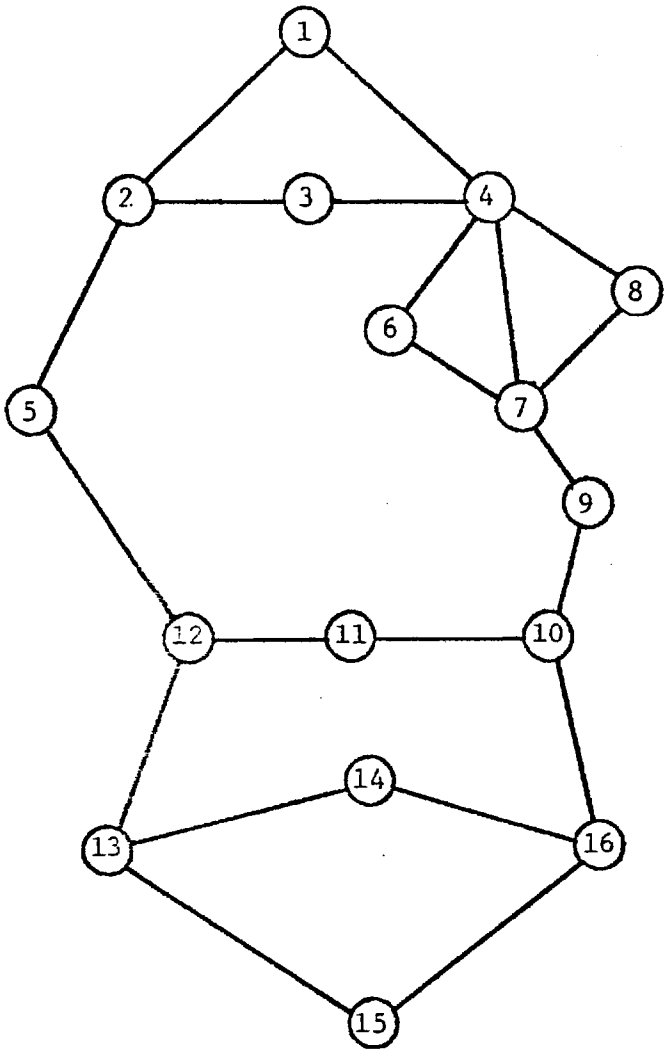


Figure 6. Graph of Example

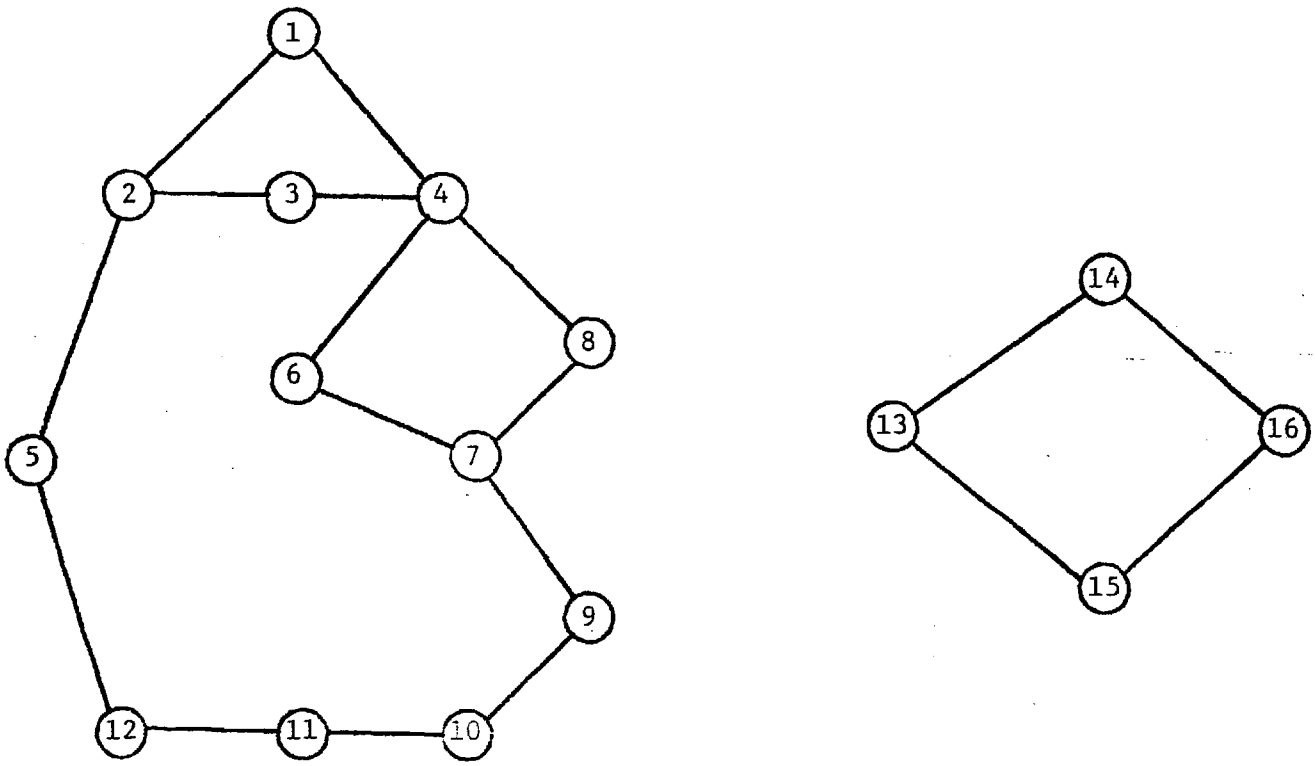


Figure 7. Blocks B_1 and B_2 defined for edges (10,16) and (12,13)

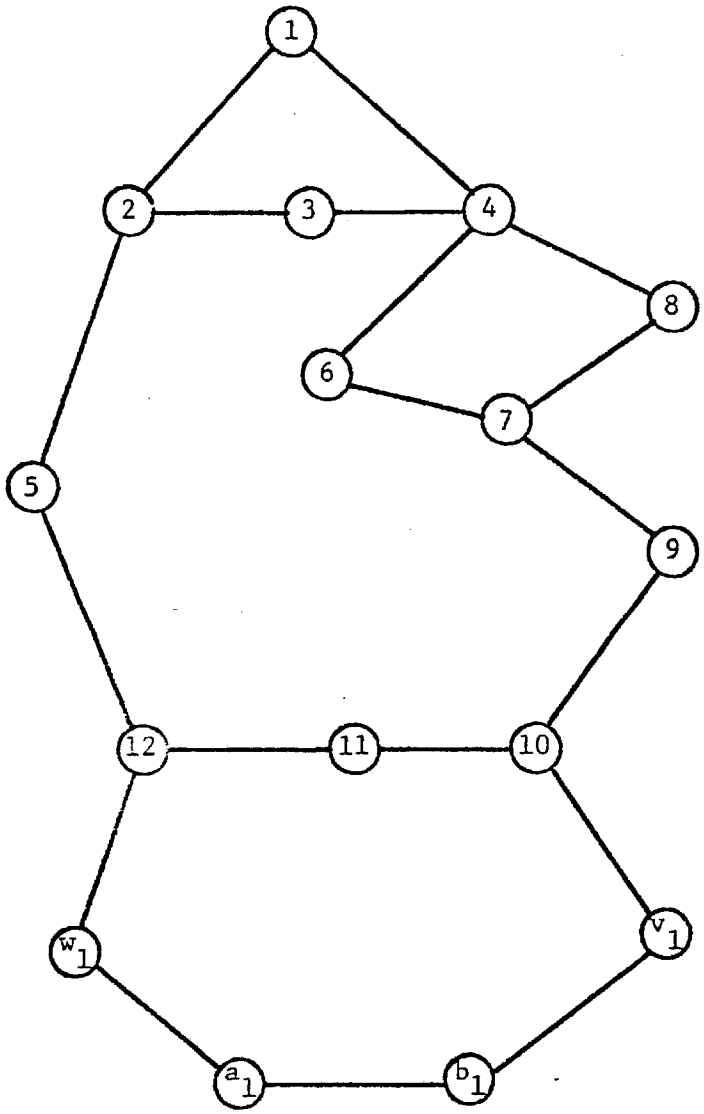


Figure 8. G^2

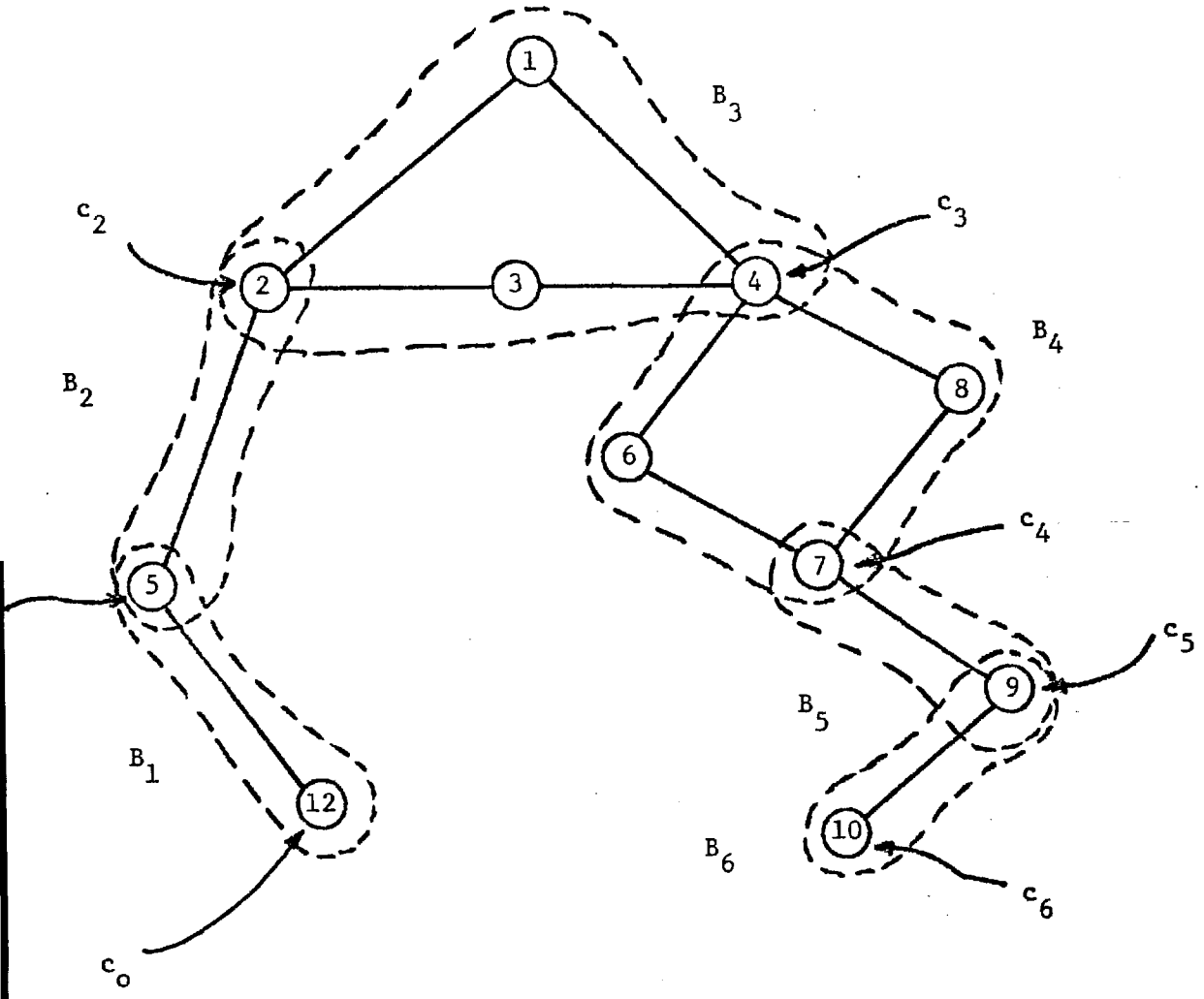
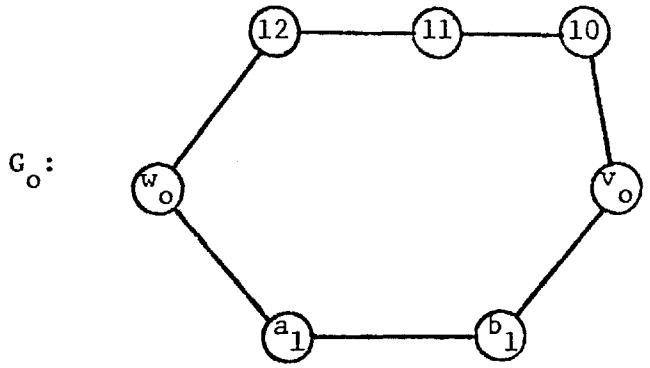
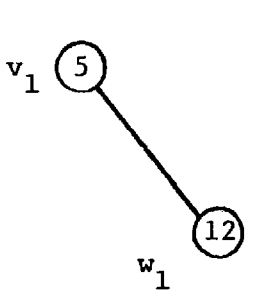
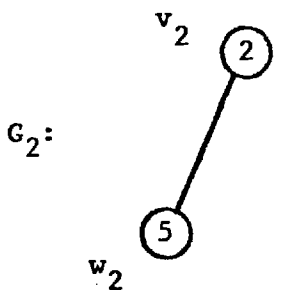


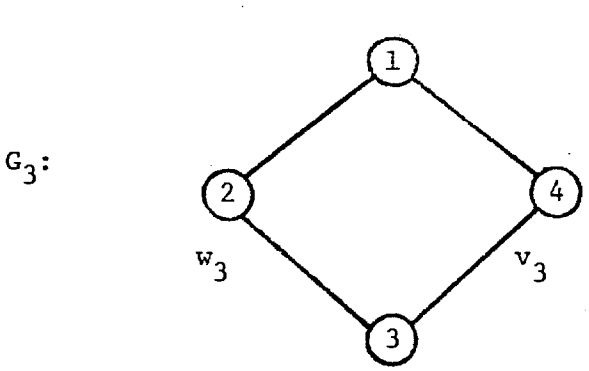
Figure 9. Blocks Induced by $G^2 \setminus E(C_1)$

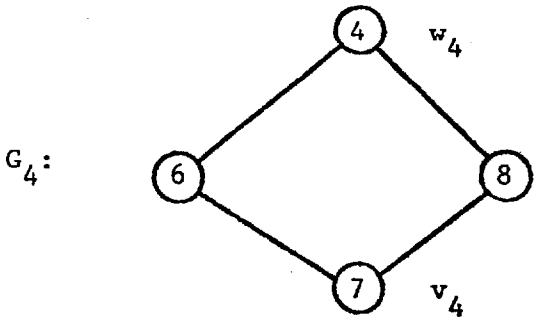






$C_o = C_1 (G_o)$





$C_3 = G_3$

$C_4 = G_4$

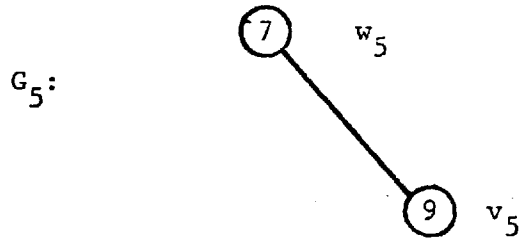


Figure 10. Unprocessed Subgraph Sets Formed by Decomposition of Step 3c(EPS)

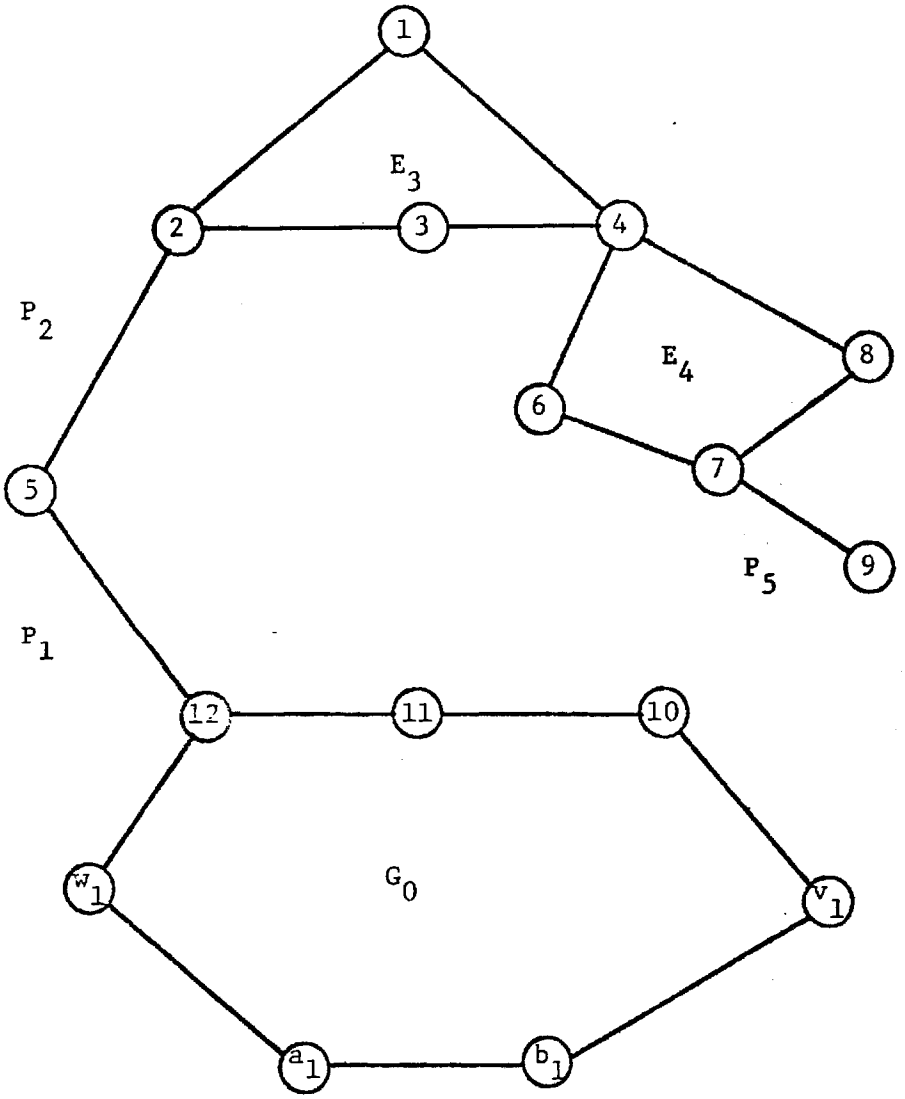


Figure 11. EP-Subgraph, S

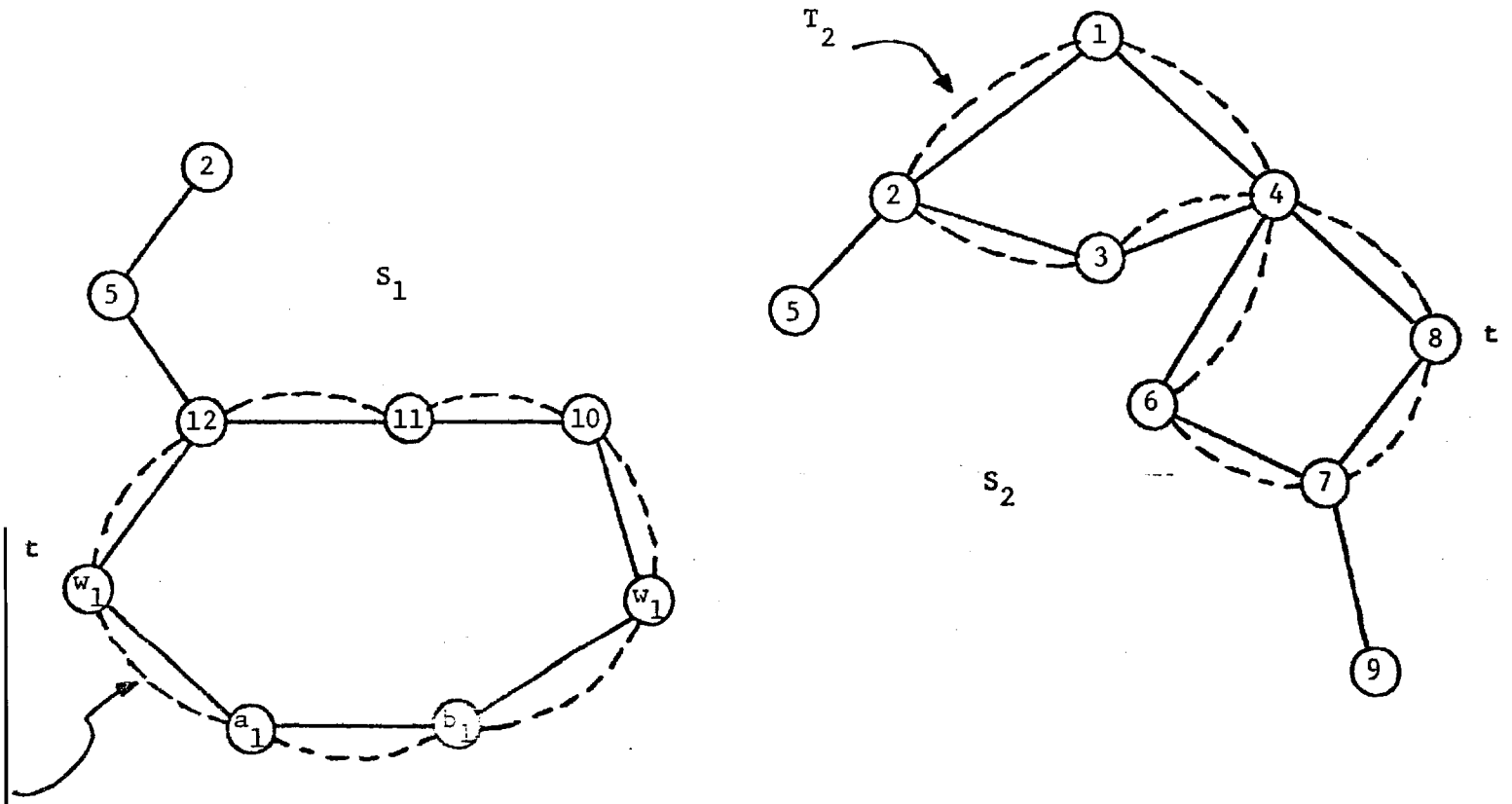


Figure 12. Subgraphs S_1 and S_2 and Eulerian Cycles T_1 and T_2

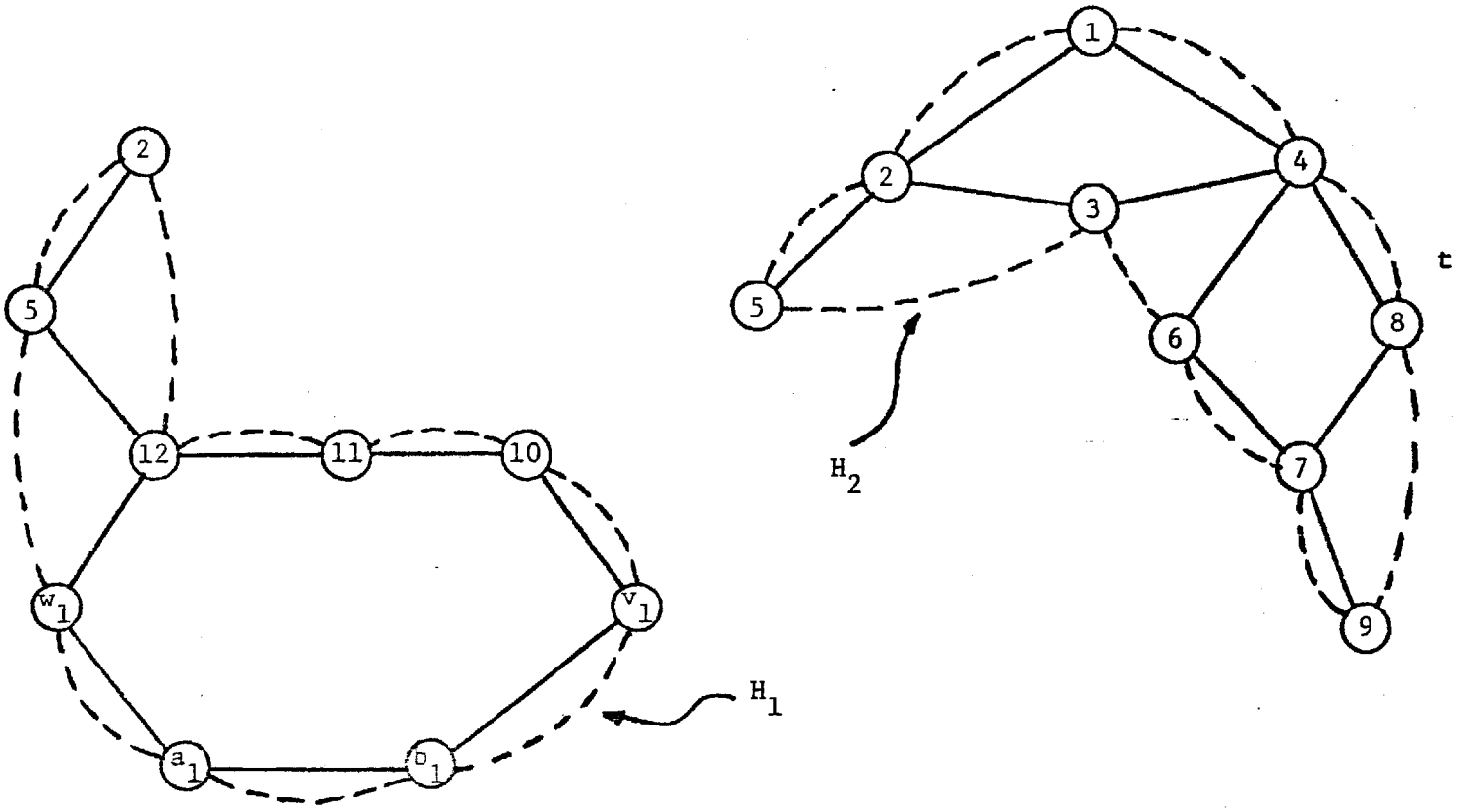


Figure 13. Hamiltonian Cycles in the Squares of S_1 and S_2

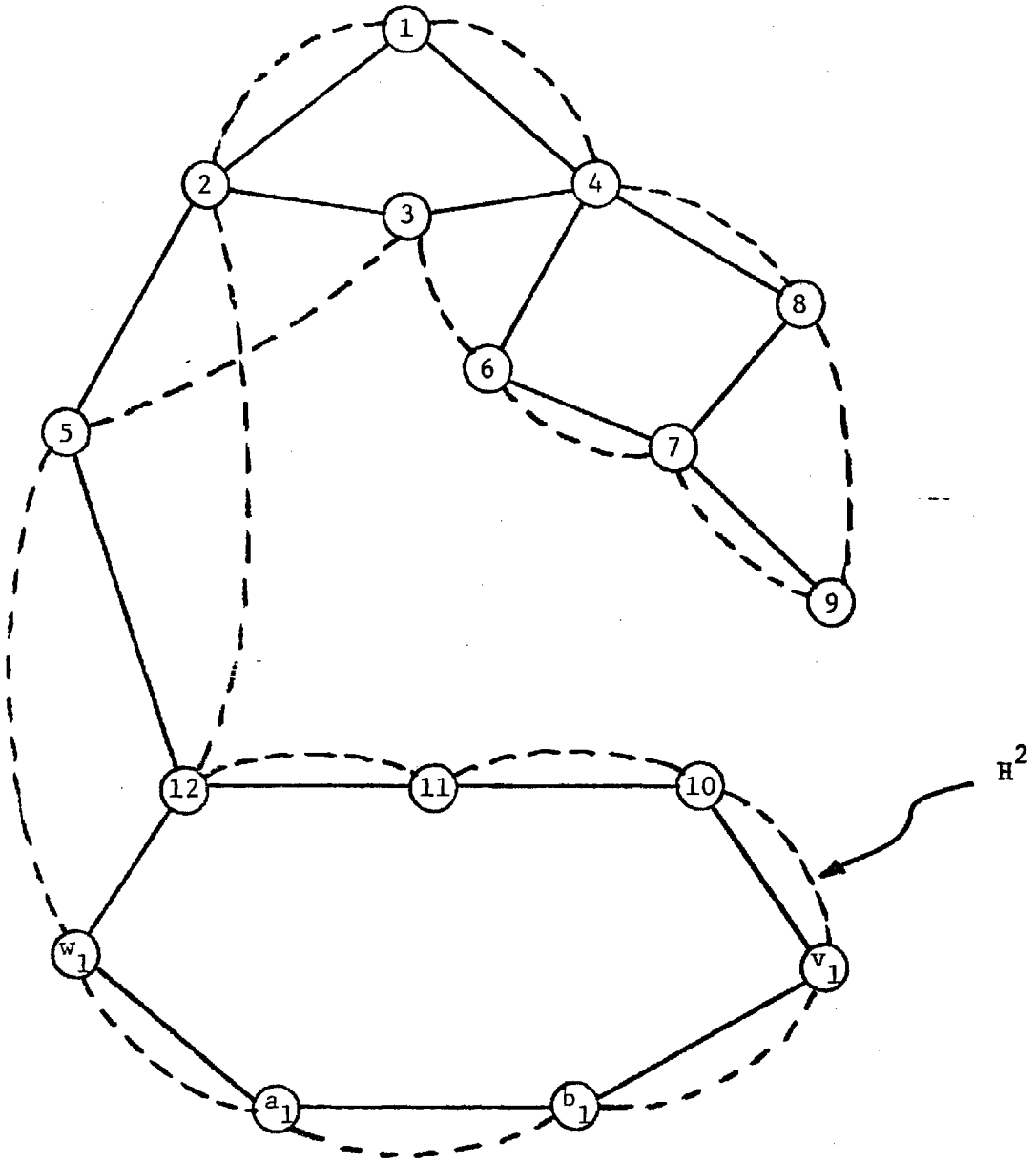


Figure 14. Hamiltonian Cycle H^2 in the Square of EP - subgraph S.

8. REFERENCES

1. Aho, A., J.E. Hopcroft, and J.D. Ullman (1976), The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA.

2. Fleischner, H. (1974a), "On Spanning Subgraphs of a Connected Bridgeless Graph and Their Application to DT-Graphs," J. Comb. Theory (B), 16, pp. 17-28.

3. Fleischner, H. (1974B), "The Square of Every 2-Connected Grasph is Hamiltonian," J. Comb. Theory (B), 16, pp. 29-34.

4. Hochbaum, D.S. and D.B. Shmoys (1983), "Best Possible Heuristics for the Bottleneck Wandering Salesperson and Bottleneck Vehicle Routing Problems," No. University of California, Berkeley, unpublished manuscript.

5. Lau, H.T. (1981), "Finding EPS-Graphs," Monatshefte für Mathematik, 92, pp. 37-40.

6. Nash-Williams, C. St. J. A. (1968), Problem No. 48, Theory of Graphics (P. Erdos and G. Kantona, Eds.) Academic Press, New York.

7. Parker, R.G. and R.R. Rardin, (1983), "Guaranteed Performance Heuristics for the Bottleneck Traveling Salesman Problem," Operations Research Letters, to appear.

8. Rosenstiehl, P. (1971), "Labryinthologie Mathematique," Math. Sci. Humaines, 33.

Industrial and Systems Engineering
Report Series J-82-4
June, 1982

DEVELOPMENT OF A PROGRESSIVE DISAGGREGATION ALGORITHM FOR FIXED CHARGE NETWORK FLOW PROBLEMS

by

Ronald L. Rardin*

and

Oscar Adaniya**

* Associate Professor, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

** Assistant Professor, Industrial Engineering, University of Miami, Box 248294, Coral Gables, Florida 33124

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS-801954

Abstract

Fixed charge network flow problems model network design and location settings by allowing both fixed and variable charges for arc flow. Recent research has shown that very close approximations to mixed-integer solutions for each problems can be obtained from massive linear programs wherein flows are artificially disaggregated into separate components for each origin - destination pair. This paper develops the strategy of a progressive disaggregation algorithm employing the latter linear programming relaxation. However, flows are initially undisaggregated. As computation proceeds, supply and demand subsets are further and further partitioned to tighten the relaxation as required without incurring the computational burden of a complete disaggregation into supply-demand pairs.

1. Introduction

The fixed charge network flow problem in one commodity is typically formulated

$$\min \sum_{(i,j)\in E} v_{ij}x_{ij} + \sum_{(i,j)\in E} f_{ij}y_{ij} \tag{1}$$

$$\text{s.t.} \sum_{(i,\beta)\in E} x_{i\beta} = d_{\beta} \quad \text{for all } \beta\in\mathcal{D} \tag{2}$$

$$\sum_{(\alpha,j)\in E} x_{\alpha j} \le s_{\alpha} \quad \text{for all } \alpha\in S \tag{3}$$

(FC)

$$\sum_{(\ell,j)\in E} x_{\ell j} - \sum_{(i,\ell)\in E} x_{i\ell} = 0 \quad \text{for all } \ell\in T \tag{4}$$

$$x_{ij}/u_{ij} \le y_{ij} \quad \text{for all } (i,j)\in E \tag{5}$$

$$x_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{6}$$

$$1 \ge y_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{7}$$

$$y_{ij} \text{ integer} \quad \text{for all } (i,j)\in E \tag{8}$$

Here E is the arc set of a specified network; x_{ij} is the flow from i to j; S, $\mathcal{D}$ and T are the supply point, demand point and transshipment point subsets of nodes respectively; s_{α} is the supply at point α; d_{β} is the demand at point β; and u_{ij} is a capacity of arc (i,j) flow. Costs (1) include a variable (per unit flow) cost v_{ij} and a fixed charge f_{ij} "switched on" by the 0-1 variable y_{ij} whenever $x_{ij} > 0$. We assume throughout that all f_{ij} and v_{ij} are nonnegative although the latter requirement can be relaxed in some cases.

Formulation (FC) gives a correct mixed-integer statement of the fixed charge network flow problem, but its linear programming relaxation, (obtained by deleting

constraint (8)) often provides only a very poor approximation to the mixed integer form. Rardin and Choe (1979) and Rardin (1982) demonstrated that a much better linear programming approximation is obtained by disaggregating flows x_{ij} into components $x_{ij}[\alpha,\beta]$ distinguished by the supply point α at which the flow originated and the demand point β to which it is defined.

Such a multi-commodity formulation is

$$\min \quad \sum_{(i,j)\in E} v_{ij} \sum_{\alpha\in S} \sum_{\beta\in D} x_{ij}[\alpha,\beta] + \sum_{(i,j)\in E} f_{ij}\, y_{ij} \tag{9}$$

$$\text{s.t.} \quad \sum_{\alpha\in S} \sum_{(i,\beta)\in E} x_{i\beta}[\alpha,\beta] = d_{\beta} \qquad \text{for all } \beta\in D \tag{10}$$

$$\sum_{\beta\in D} \sum_{(\alpha,j)\in E} x_{\alpha j}[\alpha,\beta] \le s_{\alpha} \qquad \text{for all } \alpha\in S \tag{11}$$

(MC)

$$\sum_{(\ell,j)\in E} x_{\ell j}[\alpha,\beta] - \sum_{(i,\ell)\in E} x_{i\ell}[\alpha,\beta] = 0 \qquad \text{for all } \alpha\in S,\ \beta\in D,\ \ell\in T \tag{12}$$

$$(1/u_{ij}) \sum_{\alpha\in S} \sum_{\beta\in D} x_{ij}[\alpha,\beta] \le y_{ij} \qquad \text{for all } (i,j)\in E \tag{13}$$

$$x_{ij}[\alpha,\beta] \ge 0 \qquad \text{for all } (i,j)\in E,\ \alpha\in S,\ \beta\in D \tag{14}$$

$$1 \ge y_{ij} \ge 0 \qquad \text{for all } (i,j)\in E \tag{15}$$

$$y_{ij} \text{ integer} \qquad \text{for all } (i,j)\in E \tag{16}$$

$$\frac{x_{ij}[\alpha,\beta]}{\min\{s_{\alpha},d_{\beta}\}} \le y_{ij} \qquad \text{for all } (i,j)\in E,\ \alpha\in S,\ \beta\in D \tag{17}$$

As mixed-integer programs, forms (FC) and (MC) are equivalent. However, disaggregation of (FC) flows x_{ij} into separate commodities $x_{ij}[\alpha,\beta]$ leads to a tighter linear programming relaxation in (MC) because of the new constraints (17). With $f_{ij} \geq 0$ the linear programming relaxation, say ($\overline{FC}$), of (FC) will always have an optimal solution with no slack in (5). Thus, ($\overline{FC}$) solutions incur only the fraction x_{ij}/u_{ij} of the fixed charge f_{ij} that flow x_{ij} forms of its capacity u_{ij}. Equation (13) enforces the same limit in ($\overline{MC}$), the linear programming, relaxation of (MC). However, (17) also forces y_{ij} to be as large as the fraction of any source α or sink β flow passing through (i,j). The improved linear programming relaxation follows when (as is usually the case), s_α and/or d_β are much smaller than u_{ij}.

Although providing generally much tighter linear programming approximations, the ($\overline{MC}$) form is an enormous linear program. For a case with 750 arcs, 25 supplies, 100 demands, and 125 transshipment nodes, (MC) has over 400 thousand main constraints and approximately 2.2 million variables. The dual ascent scheme proposed by Rardin and Choe (1979) exploits problem structure in a Lagrangean relaxation, (we give details below), but a typical iteration still involves shortest path problems for each (α,β) pair, and search over dual variables for all constraints (17). For the problem size just described, there would be 2500 such shortest path problems and approximately 1.9 million searchable dual variables.

However, the formulations (FC) and (MC) may be viewed as endpoints of a disaggregation continuum. Form (FC) treats all flows in a single commodity; (MC) disaggregates flows into artificial commodities for each origin - destination pair. Certainly, there are intermediate possibilities wherein flow is treated in groups, (A_k,B_k) with $A_k \subset S$, $B_k \subset D$.

In this paper we first sharpen the (MC) formulation and then develop

strategies for an algorithm exploiting a progressive disaggregation of $S \times D$ flows. The algorithm generally follows the Lagrangean relaxation philosophy of Rardin and Choe (1979), but processing begins with the undisaggregated form (FC), i.e. with one supply group $A_1 = S$ and one demand group $B_1 = D$. As computation proceeds supply and demand groups are progressively partitioned to create new artificial commodity structures. It is hoped that computational testing now underway will demonstrate such a progressive approach reduces total calculation to obtain a satisfactory approximation to an ($\overline{MC}$) optimum.

2. An Improved Formulation

Flow in our given network can be conceptualized as the rectangle of Figure 1. Sides reflect supplies and demands respectively. Formulation (FC), which uses only one commodity, views the rectangle of flows on arc (i,j) as a single unit x_{ij}. In (MC), each supply, demand cell of the rectangle is tabulated separately as $x_{ij}[\alpha,\beta]$. At disaggregation levels between these extremes, supplies and demands are grouped in a rectangle (A_k,B_k) collecting all flows from origins $\alpha\varepsilon A_k$ to destinations $\beta\varepsilon B_k$.

The analog of Rardin and Choe's (MC) constraint (17) for such a commodity (A_k,B_k) is

$$\frac{\sum_{\alpha\varepsilon A_k}\sum_{\beta\varepsilon B_k} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha\varepsilon A_k} s_\alpha, \sum_{\beta\varepsilon B_k} d_\beta\right\}} \le y_{ij} \quad \text{for all } (i,j)\varepsilon E \text{ and all } k \tag{18}$$

However, by treating supplies and demands separately we can expand the sums in the numerator and thus sharpen the relaxation.

Lemma 1: Improved Formulation. Let $x_{ij}[\alpha,\beta]$, s_α, d_β, S and D be as in formulation (MC), A_k a nonempty subset of S and B_k a nonempty subset of D. Then the following constraints are satisfied by every feasible (integer) solution to (FC)

$$\frac{\sum_{\alpha\varepsilon A_k}\sum_{\beta\varepsilon D} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha\varepsilon A_k} s_\alpha, \sum_{\beta\varepsilon D} d_\beta\right\}} \le y_{ij} \quad \text{for all } (i,j)\varepsilon E \text{ and all } k \tag{19}$$

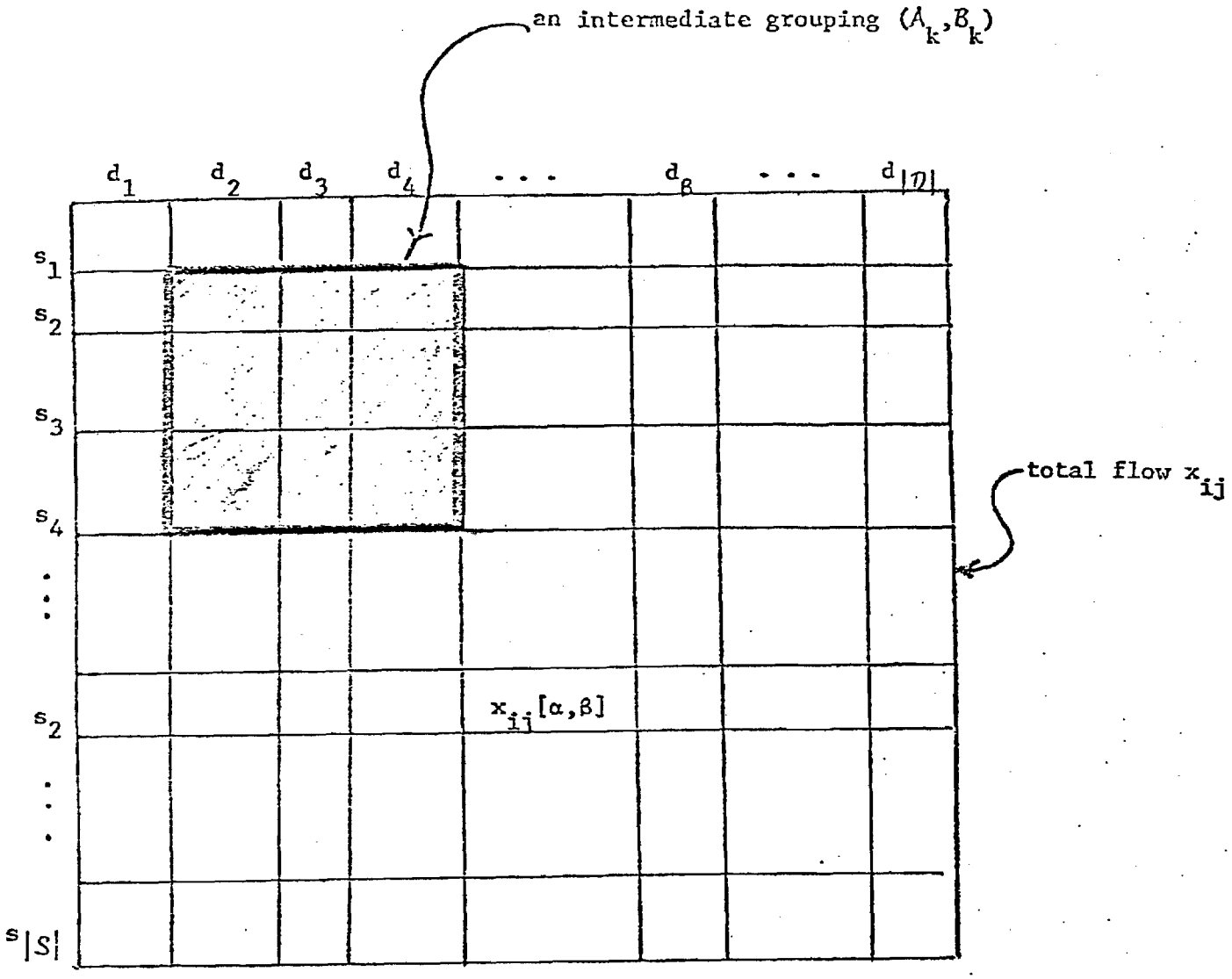


Figure 1: Total Flow as a Supplies by Demands Rectangle

$$\frac{\sum_{\alpha\in S}\sum_{\beta\in B_\ell} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\beta\in B_\ell} d_\beta,\ \sum_{\alpha\in S} s_\alpha\right\}} \le y_{ij} \quad \text{for all } (i,j)\in E \text{ and all } \ell \quad (20)$$

Furthermore, for specified $s_\alpha > 0$, $d_\beta > 0$, $x_{ij}[\alpha,\beta]$,

$$\{y_{ij} \text{ satisfying (18)}\} \supset \{y_{ij} \text{ satisfying (19) and (20)}\} \quad (21)$$

Proof: It is clear that (19) and (20) are valid in (MC); they simply require that y_{ij} be at least the fraction of supply in A_k or demand in B_ℓ passing through (i,j), respectively. To see (21) observe that if $\sum_{\alpha\in A_k} s_\alpha \le \sum_{\beta\in B_\ell} d_\beta$, (19) has the same denominator as (18), and at least as great a numerator. If $\sum_{\alpha\in A_k} s_\alpha \ge \sum_{\beta\in B_k} d_\beta$, (20) dominates (18).

∎

For a system of q commodities (A_1, B_1), (A_2, B_2),...,(A_q, B_q) there are q constraints of type (18) and potentially 2q like (19) and (20). However, any commodities k and ℓ with $A_k = A_\ell$ or $B_k = B_\ell$ have the same constraint (19) or (20) respectively. The result can be a considerable reduction in the possible number of (19) and (20). In the extreme case where every $(\alpha,\beta) \in S\times D$ forms a separate commodity, there are $|S| + |D|$ constraints (19) and (20), but $|S| \cdot |D|$ limits (18). Thus, at least, as this complete disaggregation is approached, use of (19) and (20) results in both a substantial saving of constraints and a gain in formulation tightness.

3. The Lagrangean Relaxation Setting

With even a partial disaggregation of problem flows into artificial commodities, one obtains a formidable linear program relaxation to be solved. If arc capacities (13) (or (5)) are nonbinding, Rardin and Choe (1979) showed how an effective Lagrangean relaxation of the remaining problem could be structured by summing constraints (19) and (20) in the objective function with nonnegative dual multipliers. Let $\mathcal{A} \triangleq \{A_k\}$ be the list of distinct supply subsets of current artificial commodities, $\{\sigma_{ij}[k]: A_k \in \mathcal{A}, (i,j) \in E\}$ be the nonnegative dual multipliers on corresponding constraints (19), $\mathcal{B} \triangleq \{B_\ell\}$ the list of distinct demand subsets of current commodities, and $\{\delta_{ij}[\ell]: B_\ell \in \mathcal{B}, (i,j) \in E\}$ be the dual variables on their constraints (20). Then the implied Lagrangean relaxation is as follows:

$$\min \sum_{(i,j)\in E} v_{ij} \sum_{\alpha\in S} \sum_{\beta\in \mathcal{D}} x_{ij}[\alpha,\beta] + \sum_{(i,j)\in E} f_{ij} y_{ij}$$

$$+ \sum_{(i,j)\in E} \sum_{A_k \in \mathcal{A}} \sigma_{ij}[k] \left[\frac{\sum_{\alpha\in A_k} \sum_{\beta\in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{ \sum_{\alpha\in A_k} s_\alpha, \sum_{\beta\in \mathcal{D}} d_\beta \right\}} - y_{ij} \right] \tag{22}$$

$$+ \sum_{(i,j)\in E} \sum_{B_\ell \in \mathcal{B}} \delta_{ij}[\ell] \left[\frac{\sum_{\alpha\in S} \sum_{\beta\in B_\ell} x_{ij}[\alpha,\beta]}{\min\left\{ \sum_{\beta\in B_\ell} d_\beta, \sum_{\alpha\in S} s_\alpha \right\}} - y_{ij} \right]$$

$$\text{s.t.} \sum_{\alpha\in S} \sum_{(i,\beta)\in E} x_{ij}[\alpha,\beta] = d \qquad \text{for all } \beta \in \mathcal{D} \tag{23}$$

$(P_{\sigma\delta}[A,B])$

$$\sum_{\beta\in\mathcal{D}} \sum_{(\alpha,j)\in E} x_{\alpha j}[\alpha,\beta] \le s_\alpha \quad \text{for all } \alpha\in S \tag{24}$$

$$\sum_{(\ell,j)\in E} x_{\ell j}[\alpha,\beta] - \sum_{(i,\ell)\in E} x_{i\ell}[\alpha,\beta] = 0 \quad \text{for all } \alpha\in S,\ \beta\in\mathcal{D},\ \ell\in\mathcal{T} \tag{25}$$

$$x_{ij}[\alpha,\beta] \ge 0 \quad \text{for all } (i,j)\in E,\ \alpha\in S,\ \beta\in\mathcal{D} \tag{26}$$

$$1 \ge y_{ij} \ge 0 \quad \text{for all } (i,j)\in E \tag{27}$$

$$y_{ij} \text{ integer} \quad \text{for all } (i,j)\in E \tag{28}$$

For any choice of nonnegative $\delta_{ij}[k]$ and $\sigma_{ij}[\ell]$ formulation, $(P_{\sigma\delta}[A,B])$ gives a valid lower bound on the cost of an (FC) or (MC) optimum. A search is, of course, necessary to find good dual values.

The advantage of the $(P_{\sigma\delta}[A,B])$ form lies with the fact that $[\alpha,\beta]$ systems are linked only through the objective function. Thus, for fixed dual values, $(P_{\sigma\delta}[\alpha,\beta])$ separates into a series of shortest path problems for $[\alpha,\beta]$ pairs, followed by an S to $\mathcal{D}$ transportation problem.

Including subgradient steps to improve duals and revise the present commoditization, a full procedure employing $(P_{\sigma\delta}[A,B])$ is as follows:

Step 0: Initialization. Fix dual and primal incumbent values $\nu_D^* \leftarrow -\infty$, $\nu_P^* \leftarrow +\infty$.

Step 1: Initial Disaggregation. Partition the source mode by destination node set $S\times\mathcal{D}$ into an initial series of artificial supply-demand commodities, and let A be the list of distinct supply subsets, A_k,

and $\mathcal{B}$ the corresponding list of distinct demand subsets, B_ℓ. Fix all duals $\sigma_{ij}[k]$ and $\delta_{ij}[\ell]$ at zero.

Step 2: Implicit Costs. Determine (22) objective function coefficients

$$\tilde{f}_{ij} \leftarrow f_{ij} - \sum_{A_k \in \mathcal{A}} \sigma_{ij}[k] - \sum_{B_\ell \in \mathcal{B}} \delta_{ij}[\ell] \tag{29}$$

$$\tilde{v}_{ij}[\alpha,\beta] \leftarrow v_{ij} + \sum_{\{A_k \in \mathcal{A}: \alpha \in A_k\}} (\sigma_{ij}[k]/s[k]) + \sum_{\{B_\ell \in \mathcal{B}: \beta \in B_\ell\}} (\delta_{ij}[\ell]/d[\ell]) \tag{30}$$

where

$$s[k] \triangleq \min \left\{ \sum_{\alpha \in A_k} s_\alpha, \sum_{\beta \in \mathcal{D}} d_\beta \right\} \tag{31}$$

$$d[\ell] \triangleq \min \left\{ \sum_{\beta \in B_\ell} d_\beta, \sum_{\alpha \in \mathcal{S}} s_\alpha \right\} \tag{32}$$

Step 3: Shortest Paths. For each pair (α,β) of a source and a destination node, compute the shortest path from every α to every β over arc lengths $\tilde{v}_{ij}[\alpha,\beta]$. Let $R[\alpha,\beta]$ be the set of arcs in the shortest path from node α to node β and $c[\alpha,\beta]$ its length.

Step 4: Transportation Problem. Using costs $c[\alpha,\beta]$, suppliers s_α and demand d_β, solve an $\mathcal{S}$ to $\mathcal{D}$ transportation problem. Denote by $z[\alpha,\beta]$ an optimal flow from α to β obtained in the solution to the transportation problem.

Step 5: Flow Solution: For each α and β, assign $z[\alpha,\beta]$ units of flow to all arcs (i,j) in the corresponding set of shortest path arcs $R[\alpha,\beta]$.

Step 6: 0-1 Problem. Compute relaxation optimal values for the y_{ij} variables via

$$y_{ij} \leftarrow 1 \quad \text{if } \tilde{f}_{ij} \leq 0 \text{ and } 0 \text{ otherwise.}$$

Step 7: Dual Solution. Compute a dual solution, v_D, as the sum of the costs of the optima in Steps 5 and 6. If $v_D > v_D^*$ save a new dual incumbent $v_D^* \leftarrow v_D$.

Step 8: Primal Solution. Create a feasible solution to (DC) by paying full fixed charges on any arc used in the flow of Step 5. Let v_P be its cost. If $v_P < v_P^*$, save a new primal incumbent as an approximate optimum to (FC). If not, check whether the rate of improvement in the ratio v_P^*/v_D^* is satisfactory. If so, go to Step 10.

Step 10: Dual Update. Modify duals $\sigma_{ij}[k]$, and $\delta_{ij}[\ell]$ by taking a finite step along a subgradient of the Lagrangean dual function at the current dual point and projecting to restore nonnegativity (see for example Bazaraa and Goode (1979) for details on subgradient schemes). Then return to Step 2.

Step 11: Disaggregation. Further subdivide the present artificial commoditization of $S \times D$. Add any newly created distinct supply subset A_k to $\mathcal{A}$ and pick an appropriate nonnegative starting value for corresponding dual variables $\{\sigma_{ij}[k]:(i,j)\in E\}$. Similarly, add newly created distinct demand subsets B_ℓ to $\mathcal{B}$ and choose nonnegative $\{\delta_{ij}[\ell]:(i,j)\in E\}$. Then, return to Step 2.

4. Artificial Commodity Structures

One important set of issues surrounding the implementation of the above algorithm concerns the family of artificial commodity structures employed. The algorithm is impacted by commodity structure in several ways.

- Relaxation Tightness. One aspect is the degree to which the linear programming relaxation of problem (9) - (16), (19), (20) tightly

approximates the underlying integer problem. Commodities impact relaxation tightness through the fact that there is one set of constraints (19) for each distinct supply set (i.e. each $A_k \varepsilon A$) and one set of constraints (20) for each distinct demand set (each $B_\ell \varepsilon B$). Relaxations associated with different commodity structures differ only in the limitations imposed by these constraints.

- Dual Variables. The number of dual variable sets $\{\sigma_{ij}[k]:(i,j)\varepsilon E\}$ and $\{\delta_{ij}[\ell]:(i,j)\varepsilon E\}$ which must be stored and searched over in any commoditization is also controlled by the dimension of the distinct supply and demand subset sets A and B. For each $A_k \varepsilon A$ and each $B_\ell \varepsilon B$ there is a set of constraints (19) or (20) and an associated set of dual variables.

- Shortest Path Problems. Step 3 of the algorithm calls for finding shortest paths between all supply-demand pairs. Arc lengths $\tilde{v}_{ij}[\alpha,\beta]$ for shortest path problems are as in (30). Assume, as is usually the case, that there are many fewer supply nodes than demand nodes (Symmetric arguements could be given for the opposite case). Then, noting all $\tilde{v}_{ij}[\alpha,\beta]$ are maintained nonnegative throughout processing, a version of the efficient Dijkstra (1959) algorithm should be employed to compute shortest paths. But the Dijstra algorithm can compute simultaneously the shortest path from one node to all other nodes. Thus, if $\tilde{v}_{ij}[\alpha,\beta]$ is independent of β, the Dijstra procedure needs to be invoked only once per $\alpha \varepsilon S$. However, if the $\sum(\delta_{ij}[\ell]/d[\ell])$ term of (30) creates different $\tilde{v}_{ij}[\alpha,\beta]$, the procedure must be applied once per $\alpha \varepsilon S$ and per demand subset with distinct $\tilde{v}_{ij}[\alpha,\beta]$. In total

$$|S| \cdot \begin{pmatrix} \text{number of combinations of} \\ B_\ell \varepsilon B \text{ to which any } \beta \\ \text{simultaneously belongs} \end{pmatrix} \tag{33}$$

shortest path will be required per execution of Step 3.

From the above it is clear that all impacts of artificial commodity structure are controlled by the supply subset list $A \triangleq \{A_k\}$ with each $A_k \subset S$ and the demand subset list $B \triangleq \{B_\ell\}$ with each $B_\ell \subset \mathcal{D}$. To compare possibilites, define a structure $[\tilde{A},\tilde{B}]$ to be tighter than another $[A,B]$

$$\left\{\begin{array}{l} y_{ij} \text{ satisfying (19) for } \tilde{A}_k \varepsilon \tilde{A} \\ \text{and (20) for } \tilde{B}_\ell \varepsilon \tilde{B} \end{array}\right\} \subset \left\{\begin{array}{l} y_{ij} \text{ satisfying (19) for } A_k \varepsilon A \\ \text{and (20) for } B_\ell \varepsilon B \end{array}\right\}$$

That is, $[\tilde{A},\tilde{B}]$ is tighter than $[A,B]$ if it provides at least as tight a linear programming relaxation. We can then obtain some simple dominance results.

Lemma 2: Dominance of Covering Subsets. Let $[A,B]$ be a commodity structure for flows in $S \times \mathcal{D}$, i.e. A a list of distinct nonempty subsets of S and B a similar list of subsets of $\mathcal{D}$. Also, define $\bar{A} \subset S - \bigcup_A A_k$ and $\bar{B} \subset \mathcal{D} - \bigcup_B B_k$. Then both $[A \cup \{\bar{A}\}, B]$ and $[A, B \cup \{\bar{B}\}]$ are tighter than $[A,B]$. Also, $[A \cup \{\bar{A}\}, B \cup \{\bar{B}\}]$ is tighter than either $[A \cup \{\bar{A}\}, B]$ or $[A, B \cup \{\bar{B}\}]$. That is, extending the parts of S and $\mathcal{D}$ covered by A and B tightens the formulation.

Proof: Immediate from the fact that new constraints (19) for $\bar{A}$ and/or (20) for $\bar{B}$ are added, without deleting any others.

∎

Lemma 3: Dominance of Partitioning Subsets. As above let $[A,B]$ be a commodity structure for flow in $S \times \mathcal{D}$, and pick any $A_k \varepsilon A$ such that $\sum_{\alpha \varepsilon A_k} s_\alpha \leq \sum_{\beta \varepsilon \mathcal{D}} d_\beta$ and any $B_\ell \varepsilon B$ with $\sum_{\beta \varepsilon \mathcal{D}} d_\beta \leq \sum_{\alpha \varepsilon S} s_\alpha$. Then both $[\hat{A},B]$, and $[A,\hat{B}]$ are tighter than $[A,B]$ and $[\hat{A},\hat{B}]$ is tighter than $[\hat{A},B]$ or $[A,\hat{B}]$ where

$$\hat{A} = A - \{A_k\} \cup \{A_k^i : \text{all } A_k^i \subset A_k,\ A_k^i \cap A_k^j = \phi \text{ for } i \neq j,\ \cup_i A_k^i = A_k\} \tag{34}$$

$$\hat{B} = B - \{B_\ell\} \cup \{B_\ell^i : \text{all } B_\ell^i \subset B_\ell,\ B_\ell^i \cap B_\ell^j = \phi \text{ for } i \neq j,\ \cup_i B_\ell^i = B_\ell\} \tag{35}$$

That is, replacing such A_k and B_ℓ by a partition of them yields a tighter relaxation.

Proof: We shall show only the case of $[\hat{A},B]$ tighter than $[A,B]$ where $\hat{A} = A - \{A_k\} \cup \{A_k^1\ A_k^2\}$ with $A_k^1 \subset A_k$, $A_k^2 \subset A_k$, $A_k^1 \cap A_k^2 = \phi$, and $A_k^1 \cup A_k^2 = A_k$. All other cases follow by analogous arguement for $\hat{B}$ and straightforward induction on the number of $\{A_k^i\}$ or $\{B_\ell^i\}$ respectively.

For our case the only difference in formulations $[A,B]$ and $[\hat{A},B]$ is the former contain

$$\frac{\sum_{\alpha \in A_k} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k} s_\alpha,\ \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{36}$$

versus the latter's

$$\frac{\sum_{\alpha \in A_k^1} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k^1} s_\alpha,\ \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{37}$$

and

$$\frac{\sum_{\alpha \in A_k^2} \sum_{\beta \in \mathcal{D}} x_{ij}[\alpha,\beta]}{\min\left\{\sum_{\alpha \in A_k^2} s_\alpha,\ \sum_{\beta \in \mathcal{D}} d_\beta\right\}} \leq y_{ij} \qquad \text{for all } (i,j) \in E \tag{38}$$

By the hypothesis that $\sum_{\alpha \in A_k} s_\alpha \leq \sum_{\beta \in \mathcal{D}} d_\beta$, the supply sum provides the minimum, in denominations of (36) - (38). Thus, noting A_k^1 and A_k^2 partition A_k, the proof reduces to showing

$$\max\left\{\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha},\ \frac{\sum_{\alpha\in A_k^2}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^2} s_\alpha}\right\} \geq$$

$$\left\{\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta] + \sum_{\alpha\in A_k^2}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha + \sum_{\alpha\in A_k^2} s_\alpha}\right\} \tag{39}$$

Assume the A_k^1 term provides the max on the left in (39). Then if (39) fails

$$\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha} < \frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta] + \sum_{\alpha\in A_k^2}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha + \sum_{\alpha\in A_k^2} s_\alpha}$$

Cross multiplying and simplifying leads to

$$\left[\sum_{\alpha\in A_k^2} s_\alpha\right]\left[\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta]\right] < \left[\sum_{\alpha\in A_k^1} s_\alpha\right]\left[\sum_{\alpha\in A_k^2}\sum_{\beta\in D} x_{ij}[\alpha,\beta]\right]$$

or

$$\frac{\sum_{\alpha\in A_k^1}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^1} s_\alpha} < \frac{\sum_{\alpha\in A_k^2}\sum_{\beta\in D} x_{ij}[\alpha,\beta]}{\sum_{\alpha\in A_k^2} s_\alpha}$$

Since this contradicts the assumption that A_k^1 provides the maximum in (39), we can conclude (39) holds, and the Lemma follows.

Lemma 2 makes it clear that tighter relaxations will result if we consider only commodity structures with $[A,B]$ covering $[S,\mathcal{D}]$, i.e.

$$\bigcup_{A} A_k = S \tag{40}$$

$$\bigcup_{B} B_\ell = \mathcal{D} \tag{41}$$

There could, of course, be a price in terms of dual variables and shortest path problems for demanding a cover. However, at most one new supply group $S - \bigcup_{A} A_k$, and one new demand group $\mathcal{D} - \bigcup_{B} B_\ell$, would have to be added to a non-covering $[A,B]$. Thus, only two new sets of dual variables and perhaps no new shortest path calculations are implied. For these reasons we enforce (40) and (41) in all further discussion.

We shall also demand commodity structures be nonoverlapping i.e.

$$A_k \cap A_i = \emptyset \qquad \text{for all } A_k, A_i \in A,\ i \neq k \tag{42}$$

$$B_\ell \cap B_j = \emptyset \qquad \text{for all } B_\ell, B_j \in B,\ j \neq \ell \tag{43}$$

Lemma 3 provides part of the arguement for the latter restrictions. That lemma shows relaxations are usually tightened when a supply set A_k (or a demand set B_ℓ) is partitioned. It also follows, for example, that when $A_1 \subset A_2$, it is preferable to include sets A_1 and $A_2 - A_1$ in the commodity structure instead of A_1 and A_2. We see that there is usually a gain in relaxation tightness when supply or demand sets do not overlap. In the $A_1 \subset A_2$ example there was not even an increase in dual variables. However, replacing an arbitrary A_1 and A_2 by $(A_1 - A_2)$, $(A_1 \cap A_2)$, and $(A_2 - A_1)$ would tighten the relaxation only by a net increase of one system of dual variables.

The other arguement for nonoverlapping sets as in (42) and (43) relates to the number of shortest path problems (33). Since subsets in any list B are distinct, (33) cannot be less than $|S| \cdot |B|$. Any B satisfying (43) achieves that lower limit.

5. Implementation Issues

Based on the analysis of the previous section, we propose to implement the Lagrangean relaxation algorithms of Section 3 via supply group and demand group lists A and B that always partition S and D as in (40) - (43). Step 1 will create initial partitions, and each time disaggregation Step 11 is executed either some $A_k \in A$ will be replaced by two nonoverlapping sets A_k^1 and A_k^2, or some $B_\ell \in B$ will be replaced by a similarly partitioning pair B_ℓ^1, B_ℓ^2.

Even within this approach to disaggregation, there remain many issues regarding implementation of the algorithm of Section 3. When the algorithm starts, a decision must be made with regard to the initial number of subsets in A and B and the elements of each of these subsets. Then, at every iteration it must be decided whether to further the disaggregation by partitioning an $A_k \in A$ or $B_\ell \in B$. When the decision to proceed with the disaggregation is made, a series of additional decisions are confronted, including selection of the subset to be partitioned, the assignment of its elements to the new subsets, and the initialization of the dual variables corresponding to the new subsets.

5.1 Initial Generation of Artificial Commodities

At the start of the procedure it could be decided to have one or more elements in partitioning lists A and B. If the decision is to start with singletons $A = \{S\}$, $B = \{D\}$, all further partitioning of the original source node set and the original destination node set will be performed in the disaggregation Step 11.

An alternative is to partially partition S and D from the beginning. In general more dual variables and more shortest path problems will result in early iteration. However, if the source nodes and the destination nodes are initially grouped based on a careful analysis of the problem to be solved, the relaxation may be much tighter so that progress on the dual bound in the initial stages of the procedure is faster, favorably compensating the additional computational burden brought on by handling more artificial commodities from the beginning. It is also possible that by starting from an "intelligent" list of supply and demand subsets, further disaggregation of these initial subsets would be more beneficial because the initial grouping has already considered concerns too bulky to include each disaggregation step. Finally, an initial subdivision of S and D obviously implies the number of times the disaggregation step will later be invoked by the algorithm is significantly reduced. Thus results may be less sensitive to the effectiveness and efficiency of Step 11 calculations.

In light of these potential advantages non-singleton initial disaggregations are being tried in computational testing presently underway. In picking initial groups the goal is to quickly reach a tight relaxation without producing too many elements of the initial A and B lists. Noting the form of constraints (19) and (20) it appears we would like to segregate supply and demand points with large s_α and d_β respectively. Otherwise, their presence in the denominator of (19) or (20) dilutes the impact of other flows on y_{ij}. Similarly, if a node is isolated, and thus particularly expensive to service, it seems reasonable to employ a strong relaxation in regard to it, i.e. isolate it in a separate supply or demand set.

For these reasons the initial disaggregation Step 1 being tested automatically segregates in one-point sets any supplies and demands with unusually high servicing cost or supply/demand. For remaining supply and demand points,

constraints (19) and (20) will be strongest if flows tending to have a common shortest supply-demand path are grouped. In the algorithms initial groups are formed so that ones with the most common path elements are together.

Figure 2 shows a single-supply example of these initial disaggregation notions. Since there is only one supply, $A = S = \{1\}$. The initialization rules we have outlined would create a starting partition of $\mathcal{D} = \{2,4,5,6,7,8\}$ as $B = \{\{5\}, \{6\}, \{2,4\}, \{7,8\}\}$. Node 5 is isolated because of its high demand, node 6 because arcs entering it are particularly costly. Among the remaining nodes, 2 is placed with 4 because all paths to 4 pass through 2, and 7 with 8 because many paths to 8 transit 7.

5.2 Selections of the New Partition

In the dual ascent procedure, used in conjunction with the progressive disaggregation procedure described herein, whenever the rate of improvement on the bound of the optimal solution to $(P_{\sigma\delta}[A,B])$ does not meet the minimum standards set beforehand, it signals the need to further disaggregate some of the current artificial commodities. This is carried out by partitioning one or more supply and/or destination node subsets. As noted above we have chosen to partition only one subset at any one time. The main reason for such choice is to keep the procedure simple while still achieving the goals of the disaggregation.

The selection of the subset to be partitioned involves ranking the current subsets according to some criterion that matches our strategic objective — significiant improvement of the dual bound. As we have explained earlier, the disaggregation pattern affects the dual bound only through constraints (19) and (20). In the algorithm of Section 3, those constraints are included in the $(P_{\sigma\delta}[A,B])$ objective function as terms

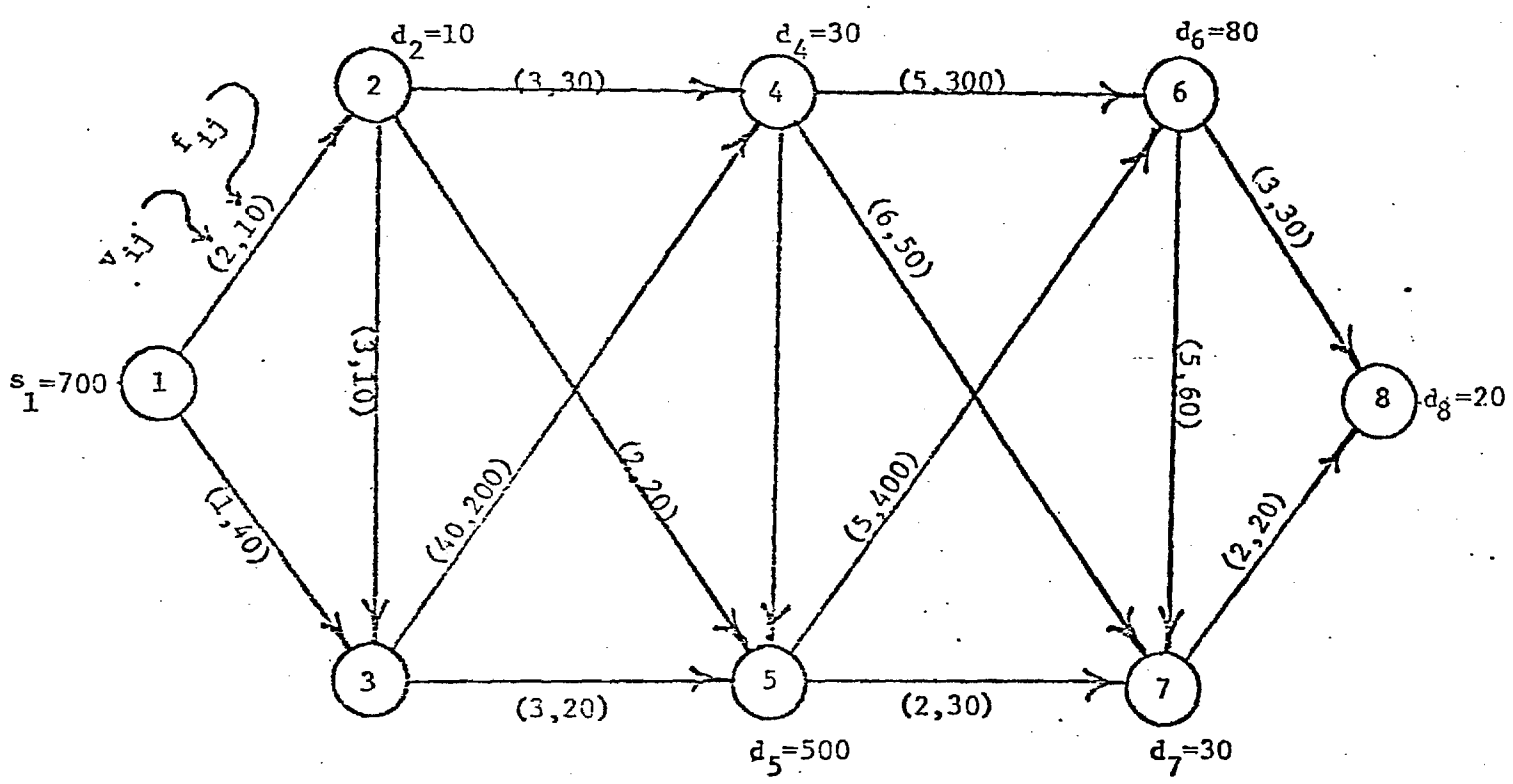


Figure 2. Initial Disaggregation Example

$$\sum_{(i,j)\in E} \sigma_{ij}[k]\left[\left(\frac{f_{ij}}{n[k]}\right)\left(\frac{\sum_{\alpha\in A_k}\ \sum_{\beta\in D} x_{ij}[\alpha,\beta]}{s[k]}\right) - y_{ij}\right] \tag{44}$$

and

$$\sum_{(i,j)\in E} \delta_{ij}[\ell]\left[\left(\frac{f_{ij}}{n[\ell]}\right)\left(\frac{\sum_{\beta\in B_\ell}\ \sum_{\alpha\in S} x_{ij}[\alpha,\beta]}{d[\ell]}\right) - y_{ij}\right] \tag{45}$$

where $s[k]$ and $d[\ell]$ are as in (31) and (32). One new element is nonnegative weights ($f_{ij}/m[k]$) and ($f_{ij}/n[\ell]$) used to scale constraints (19) and (20) for greater subgradient search efficiency. Generally, $m[k]$ is similar in magnitude to $s[k]$, and $n[\ell]$ to $d[\ell]$.

Since the expressions in (44) and (45) are less than or equal to zero in feasible solutions, minimizing their absolute value will tend to improve the dual bound quality. Consequently, we select for partition the subset for which the corresponding expression (44) or (45) is the most negative. The implementation of this selection rule is very simple and it does not involve any additional calculations, since the values of expressions (44) and (45) are always readily available in the dual ascent procedure where they are used in evaluating the objective function.

Once the subset to be partitioned is identified, it is necessary to determine how to partition it. This includes deciding how many new subsets to create and which elements of the subset being partitioned to assign to the new subsets.

With regard to the composition of the two new subsets, a criterion similar to the one used in selecting the subset to be partitioned is applied. For each element of the selected subset, its contribution to the expression in (44) for a source node subset, or to (45) for a destination node subset, is evaluated. Based on these contributions, the elements with the highest contributions will be assigned to one of the subsets, and the rest of the elements will be assigned to the other. Each of the new subsets is required to have the same number of elements, so that all singletons will be reached in the minimum number of partitions. Again, these decision rules are quite simple to implement because (44) and (45) are readily available.

5.3 Initializing Dual Variables

Once it has been decided to partition a supply group A_k or demand group B_ℓ, initial values must be chosen for dual varialbes $\sigma_{ij}[k]$ or $\delta_{ij}[\ell]$ and for scaling coefficients m[k] or n[ℓ]. We shall discuss the case of partitioning a demand set B_ℓ into two new sets B_p and B_q for which we seek new duals $\{\delta_{ij}[p]$ and $\delta_{ij}[q]$: $(i,j)\in E\}$ and scaling weights n[p] and n[q]. The case of partitioning a supply subset A_k is completely analogous.

In the previous section we showed how the goal in selecting B_p and B_q was one of maximizing the short term improvement in dual bound. We would, of course, like initial dual variables to also advance the dual solution. But there is another important issue: we desire stability in the dual search so that any poorly chosen duals will quickly be corrected by Step 10 of the algorithm.

To obtain stability, we seek to assure that the x and y primal solutions of Steps 5 and 6 of the algorithm (Section 3) will not decrease violently in the first iteration after disaggregation. (If group selection was sound the dual value should improve).

At Step 6, $y_{ij} \leftarrow 1$ if $\hat{f}_{ij} \leq 0$ and 0 otherwise, where (including the scaling factor $f_{ij}/n[\ell]$)

$$\tilde{f}_{ij} = f_{ij}\left[1 - \sum_{A_k \in A} \sigma_{ij}[k]/m[k] - \sum_{B_\ell \in B} \delta_{ij}[\ell]/n[\ell]\right] \tag{46}$$

Dividing B_ℓ into B_p and B_q in the B list will merely replace

$$\frac{\delta_{ij}[\ell]}{n[\ell]} \text{ with } \frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]}$$

Thus, the y_{ij} solution will be unchanged if

$$\frac{\delta_{ij}[\ell]}{n[\ell]} = \frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]} \tag{47}$$

To similarly preserve the $x_{ij}[\alpha,\beta]$ solution of algorithm Steps 3-5, we desire to leave unchanged shortest path arc lengths

$$\tilde{v}_{ij}[\alpha,\beta] = v_{ij} + f_{ij}\left[\sum_{\{A_k \in A:\alpha \in A_k\}} \frac{\sigma_{ij}[k]}{m[k]s[k]} + \sum_{\{B_\ell \in B:\beta \in B_\ell\}} \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}\right] \tag{48}$$

After partition each $ will belong to B_p or B_q, but not both. Thus, either

$\frac{\delta_{ij}[p]}{n[p]d[p]}$ or $\frac{\delta_{ij}[q]}{n[q]d[q]}$ will replace $\frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$ in (48),

The dual selection we propose is fixing

$$n[p] \leftarrow n[\ell] \tag{49}$$

$$n[q] \leftarrow n[\ell] \tag{50}$$

$$\hat{\delta}_{ij}[p] \leftarrow \delta_{ij}[\ell] \frac{d[p]}{d[\ell]} \tag{51}$$

$$\hat{\delta}_{ij}[q] \leftarrow \delta_{ij}[\ell] \frac{d[q]}{d[\ell]} \tag{52}$$

Substitution in (47) gives

$$\frac{\delta_{ij}[p]}{n[p]} + \frac{\delta_{ij}[q]}{n[q]} = \frac{\delta_{ij}[\ell]}{n[\ell]} \left[\frac{d[p]+d[q]}{d[\ell]}\right] = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

the last because B_p and B_q partition demands in B_ℓ. Also, (49)-(52) yield

$$\frac{\delta_{ij}[p]}{n[p]d[p]} = \frac{\delta_{ij}[\ell]}{n[\ell]d[p]} \left(\frac{d[p]}{d[\ell]}\right) = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

and

$$\frac{\delta_{ij}[q]}{n[q]d[q]} = \frac{\delta_{ij}[\ell]}{n[\ell]d[q]} \left(\frac{d[q]}{d[\ell]}\right) = \frac{\delta_{ij}[\ell]}{n[\ell]d[\ell]}$$

as required to preserve the $\bar{v}_{ij}[\alpha,\beta]$ of (48).

6. Experimentation

Previous sections outlined the development of a strategy for implementing progressive disaggregation in the context of a Lagrangean relaxation algorithm for tight formulations of fixed charge network flow problems. Justifications provided for details of the algorithm do consider problem properties, but their true effectiveness can only be measured empirically. Thus, a series of experiments involving variants of these strategic decisions is presently underway.

REFERENCES

Bazaraa, M.S. and J.J. Coode (1979) "A Survey of Various Tactics for Generating Lagrangian Multipliers in the Context of Lagrangian Duality," European Journal of Operations Research, 3, 322-338.

Dijkstra, E.W. (1959), "A Note on Two Problems in Connexion with Graphs," Numer Math, 1, 269-271.

Rardin, R.L. (1982), "Tight Relaxations of Fixed Charge Network Flow Problems," Industrial and Systems Engineering Report Series No. J-82-3, January.

Rardin, R.L. and Ui Choe (1979), "Tighter Relaxations of Fixed Charge Network Flow Problems," Industrial and Systems Engineering Report Series J-79-18, May.

Industrial Systems and Engineering
Report Series No. J-82-3
January, 1982

TIGHT RELAXATIONS OF FIXED CHARGE NETWORK FLOW PROBLEMS

by

Ronald L. Rardin*

* Associate Professor, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332

Abstract: A vast number of important engineering and management problems can be viewed as network flow problems with fixed charges for opening arcs. This research derives new, tight, linear programming relaxations for such problems based on a disaggregation of flows. The concenpt behind such relaxations is presented, and an algorithm for their solution is discussed.

This material is based upon work partially supported by the National Science Foundation under Grant Number ECS-801954.

1. Introduction

A vast number of important engineering and management problems in distribution, communication, transportation, and facilities location can be viewed as single or multi-commodity network flow problems with fixed charges for constructing/setting up/installing arcs. Such problems with commodities in P can be stated in mixed-integer form as follows:

$$\min \quad \sum_{p \in P} v^p x^p + fy \tag{1}$$

(MFP) s.t.

$$Ex^p = b^p \qquad \text{for all } p \in P \tag{2}$$

$$x^p \geq 0 \qquad \text{for all } p \in P \tag{3}$$

$$(1/u_j) \sum_{p \in P} x_j^p \leq y_j \qquad \text{for all } j \in A \tag{4}$$

$$0 \leq y \leq 1 \tag{5}$$

$$y \text{ integer} \tag{6}$$

Here E is the vertex-arc incidence matrix of a directed graph, $G(V,A)$, x^p is the flow of commodity p on that network, v^p is the variable (per unit) cost of such flow, b^p is a requirements vector for commodity p (having components summing to zero), u_j is the capacity of arc j of A, f_j is the fixed charge on arc j, and y_j is a 0-1 variable switching "on" the fixed charge when flow through arc j is allowed. I assume throughout that all f_j are nonnegative. If capacities, u_j, are not naturally apparent in the problem setting, they can usually be generated as any number greater than or equal to the maximum flow through the arc.

Figure 1 shows a simple numerical example with $|P| = 1$ commodity. All 10 units of flow originate at vertex 1; 5 are required at vertex 3 and 5 at vertex 4. It is easy to check that an optimal solution sends one unit 1-4, 4 units

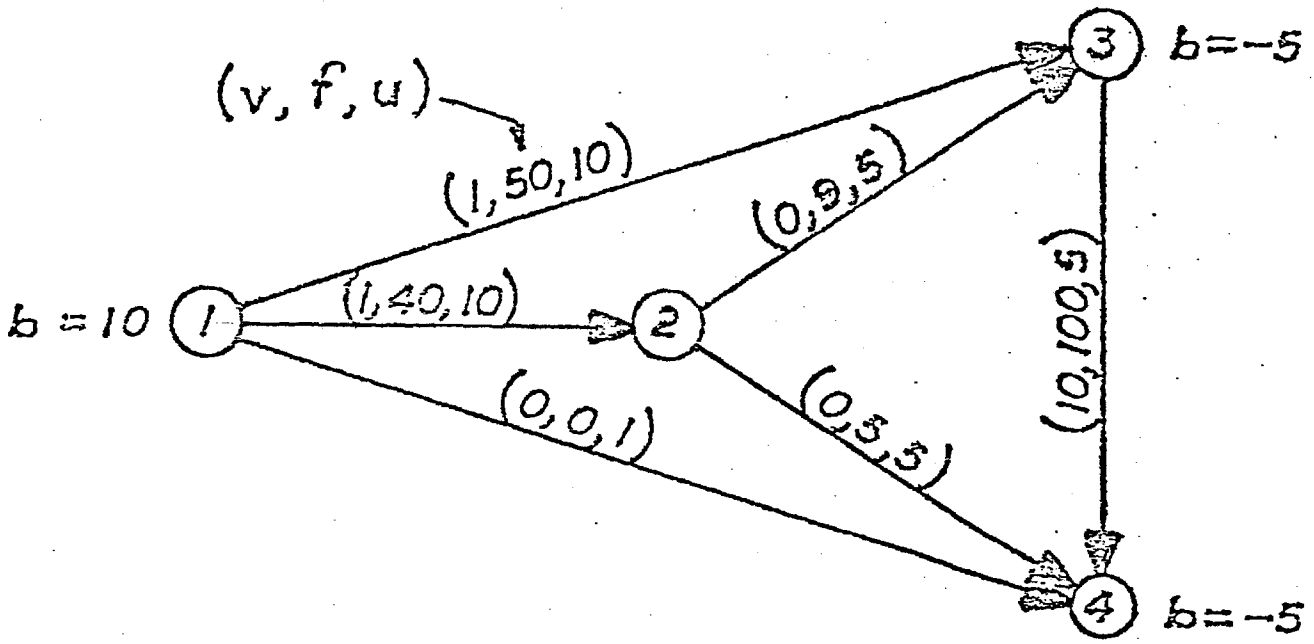


Figure 1. An Example Network

1-2-4, and 5 units 1-2-3. Total cost is 63.

2. The Standard Relaxation

Since the early work of Balinski [1, 2] a standard approach to dealing with problems (MFP) has been to solve linear programming relaxations ($\overline{\text{MFP}}$) obtained when constraints (6) are dropped. Such relaxations provide both bounds for branch-and-bound schemes and a source of approximate solutions; given an ($\overline{\text{MFP}}$) optimum, one need only round "up" all positive, but fractional y_j to obtain a feasible solution to (MFP).

For the above example this ($\overline{\text{MFP}}$) relaxation is solved by sending 1 unit 1-4, 4 units 1-2-4, and 5 units 1-3; total cost is 54 (83% of the optimal 63). When rounded "up" this solution costs 104 (165% of optimal).

Neither of these values is very satisfactory, and actual experience is often much worse. To see why, observe that the constraints (4) will always be tight in some optimal solution for ($\overline{\text{MFP}}$); where $f_j > 0$, slack in such constraints implies unnecessary cost. Since constraints (4) will be equalities in ($\overline{\text{MFP}}$), their effect is to prorate the fixed cost, f_j, over the corresponding capacity. For example, in arc (1-2) of Figure 1, 4/10 of the fixed cost, 40, would be paid in the ($\overline{\text{MFP}}$) optimum because 4/10 of the capacity, 10, is used by the optimal flow. If capacities are large, it is easy to see that this proration process would soon negate, or nearly negate, the impact of fixed costs on ($\overline{\text{MFP}}$) optima. Optimal relaxation solutions tend to use many arcs at relatively small fractions of capacity. This is particularly so when capacities are artificially created as maximum possible flows.

3. A Disaggregated Formulation

In a number of special cases, including warehouse location problems (Balinski [2], Davis and Ray [4], Erlenkotter [5], Bilde and Krarup [3], Geoffrion and Graves [7]) and uncapacitated problems (Magnanti and Wong [8]), various researchers have

shown the merit of disaggregating (MFP) flows to obtain linear programming relaxations that more closely approximate the mixed-integer problems. One can retrieve these special cases and extend the notion to all (MFP) by recognizing that flow in any commodity can always be disaggregated into separate commodity flows between origin-destination pairs of the requirements vector, b^p. Specifically, let $x^p[s,t]$ be a vector showing the flow of the portion of commodity p originating at source s and destined for sink t. Then an equivalent mixed-integer form to (MFP) is

$$\min \sum_{p\in P} \left(\sum_{s\in S_p} \sum_{t\in T_p} x^p[s,t] \right) + f\, w \qquad (7)$$

$$\text{s.t. } E^p x^p[s,t] = 0 \quad \text{for all } p\in P,\ s\in S_p,\ t\in T_p \qquad (8)$$

$$\sum_{t\in T_p} \sum_{\{j \text{ leaving } s\}} x_j[s,t] = b^p_s \quad \text{for all } p\in P,\ s\in S_p \qquad (9)$$

$$-\sum_{s\in S_p} \sum_{\{j \text{ entering } t\}} x_j[s,t] = b^p_t \quad \text{for all } p\in P,\ t\in T_p \qquad (10)$$

$$\text{(DFP)} \qquad x^p[s,t] \geq 0 \quad \text{for all } p\in P,\ s\in S_p,\ t\in T_p \qquad (11)$$

$$(1/u_j) \sum_{p\in P} \sum_{s\in S_p} \sum_{t\in T_p} x^p_j[s,t] \leq w_j \quad \text{for all } j\in A \qquad (12)$$

$$(1/-b^p_t) \sum_{s\in S_p} x^p_j[s,t] \leq w_j \quad \text{for all } j\in A,\ p\in P,\ t\in T_p \qquad (13)$$

$$(1/b_s^p) \sum_{t \in T_p} x_j^p[s,t] \leq w_j \quad \text{for all } j \in A,\ p \in P,\ s \in S_p \tag{14}$$

$$0 \leq w \leq 1 \tag{15}$$

$$w \text{ integer} \tag{16}$$

Here S_p = {sources for commodity p} = $\{s: b_s^p > 0\}$

T_p = {sinks for commodity p} = $\{t: -b_t^p > 0\}$

E^p = the row submatrix of E containing row $i \in \{i: b_i^p = 0\}$

In this new form w corresponds directly to y of (MFP), and flow variables are related by

$$x_j^p = \sum_{s \in S_p} \sum_{t \in T_p} x_j^p[s,t] \tag{17}$$

Relaxations (7), (8) through (10), (11), (12), (15), and (16) of (DFP) correspond to (1), (2), (3), (4), (5), and (6) of (MFP), respectively. Denote by $\nu(\cdot)$ the value of an optimal solution to problem $(\cdot)$ and by $(\overline{DFP})$ the linear programming relaxation of (DFP). Then this correspondence and the fact that (DFP) and $(\overline{DFP})$ have extra constraints (13) and (14) lead to the following conclusion:

Proposition 1. Solution values for (MFP), (DFP), $(\overline{MFP})$ and $(\overline{DFP})$ satisfy

$$\nu(\overline{MFP}) \leq \nu(\overline{DFP}) \leq \nu(DFP) = \nu(MFP) \tag{18}$$

□

The new elements in the (DFP) formulation are systems (13) and (14). Intuitively, (13) requires that , w_j, the portion of the fixed charge paid on arc j, must equal or exceed the fraction of a demand satisfied through arc j.

Similarly, (14) forces w_j to match the portions of each supply directed through arc j. The extra constraints are implied by (12) when integrality, (16), is enforced. But they may significantly improve the linear programming relaxation ($\overline{DFP}$) because f_j is now prorated over both u_j and all relevant supplies and demands. The latter are often much smaller than capacities.

The example of Figure 1 illustrates. An optimal solution to the linear programming relaxation ($\overline{DFP}$) sends 1 unit 1-4, 4 units 1-2-4, and 5 units 1-2-3. The relaxation cost is 62, 98% of the optimal 63. When all fractional w_j in the relaxation are rounded "up", a feasible solution is obtained that is indeed the (DFP) optimum. The effect of the disaggregation is seen on arc (1,2). The ($\overline{DFP}$) optimum pays the entire fixed charge, 40, because all demand at vertex 3 is satisfied through (1,2). From this example we may draw the further conclusion:

Proposition 2: In selected problems both inequalities of (18) may be strict.

4. Solving the Tighter Relaxation

If the strength of the ($\overline{DFP}$) relaxation is to be realized, an approach must be found for solving or nearly solving that massive linear program. Three cases can be identified. Uncapacitated cases have neither binding arc capacities, u_j, nor limits on supply at sources. Equivalently they are problems where constraints (12) are unnecessary and each requirements vector has only one positive component at the commodity's single source. Weakly capacitated cases admit supply limits, but do not have binding arc capacities. They include the capacitated warehouse location problem. Finally, fully capacitated problems have binding arc capacities, and possibly also binding supplies.

In both the uncapacitated and the weakly capacitated cases we can ignore constraints (12) of (DFP). Suppose we "dualize" (13) and (14), i.e. place them

in the objective function with nonnegative dual multipliers $\delta_j^p[t]$ and $\sigma_j^p[s]$, respectively, to obtain

$$\min \sum_{p\in P} v^p \left[\sum_{s\in S_p} \sum_{t\in T_p} x^p[s,t]\right] + fw$$

$$(DFP_{\delta\sigma}) \quad + \sum_{p\in P} \sum_{j\in A} \sum_{t\in T_p} \delta_j^p[t] \left[\frac{1}{-b_t^p} \sum_{s\in S_p} x_j^p[s,t] - w_j\right] \tag{19}$$

$$+ \sum_{p\in P} \sum_{j\in A} \sum_{s\in S_p} \sigma_j^p[t] \left[\frac{1}{b_s^p} \sum_{t\in T_p} x_j^p[s,t] - w_j\right]$$

s.t. (8), (9), (10), (11), (15) and (16)

For fixed δ and σ variables in $(DFP_{\delta\sigma})$ the commodities are linked only at sources and sinks (through (9) and (10)). Moreover, each origin-destination commodity problem is essentially one of picking a single path along which to ship from source to sink. Thus, one can approach $(\overline{DFP})$ by trying to maximize $v(DFP_{\delta\sigma})$ over nonnegative values of the dual variables as follows:

Step 0: Initialization. Set all $\delta_j^t[t]$ and $\sigma_j^p[s]$ to zero, and fix dual and primal incumbent solution values $v_D^* \leftarrow -\infty$, $v_P^* \leftarrow +\infty$.

Step 1: Implicit Costs. Determine (19) objective function coefficients

$$\tilde{f}_j \leftarrow f_j - \sum_{p\in P} \sum_{s\in S_p} \sigma_j^p[s] - \sum_{p\in P} \sum_{t\in T_p} \delta_j^p[t] \tag{20}$$

$$\tilde{v}_j^p[s,t] \leftarrow v_j^p + \sigma_j^p[s]/b_s^p + \delta_j^p[t]/(-b_t^p) \tag{21}$$

Step 2: Shortest Paths. For each $p\in P$, $s\in S_p$, $t\in T_p$ compute the shortest path from s to t over arc lengths $\tilde{v}_j^p[s,t]$. Let $R^p[s,t]$ be the set of arcs in that path and $c^p[s,t]$ its length.

Step 3: Transportation Problems: For each commodity $p \in P$, solve a transportation problem from sources $s \in S_p$ to sinks $t \in T_p$ with costs $c^p[s,t]$. Supplies are $\{b^p_s > 0\}$ and demands $\{-b^p_t > 0\}$. Denote by $z^p[s,t]$ an optimal flow from s to t in that transportation problem.

Step 4: Flow Solution: Construct and optimal flow for $(DFP_{\delta\sigma})$ by assigning for each p, $s \in S_p$, $t \in T_p$, $z^p[s,t]$ units of flow along all arcs in the corresponding shortest path $R^p[s,t]$.

Step 5: 0-1 Problem. Compute relaxation optimal values for the w_j variables via

$$\tilde{w}_j \leftarrow 1 \text{ if } \tilde{f}_j \leq 0 \text{ and } 0 \text{ otherwise.}$$

Step 6: Dual Solution. Compute a dual solution, v_D, as the sum of the $(DFP_{\delta\sigma})$ costs of the optima in Steps 4 and 5. If $v_D > v^*_D$, save a new dual incumbent $v^*_D \leftarrow v_D$.

Step 7: Primal Solution. Create a feasible solution to (DFP) by paying full fixed charges on any arc used in the flow of Step 4. Let v_P be its cost, and if $v_P < v^*_P$, save a new primal incumbent $v^*_P \leftarrow v_P$.

Step 8: Dual Update. If v^*_P is sufficiently close to v^*_D, stop and accept the primal incumbent as an approximate (DFP) optimum. If not, modify duals $\delta^k_j[t]$ and $\sigma^t_j[s]$ by taking a finite step along a subgradient of the function $v(DFP_{\delta\sigma})$ at the current dual point. Then return to Step 1.

Since every problem $(DFP_{\delta\sigma})$ is a Lagrangean relaxation of (DFP) (see Fisher [6] for details of such relaxations and subgradient search), and every flow of Step 4 is primal feasible we have:

Proposition 3: At any stage of the above algorithm

$$v^*_D \leq v(DFP) \leq v^*_P. \tag{22}$$

5. Preliminary Computational Experience

To see whether values in (22) could be brought close enough together to solve problems without the need for branch and bound, 15 random test problems were generated and approximately solved by the above algorithm. The problems were uncapacitated, 1-true-commodity cases with relatively high fixed charges on all arcs.

Table 1 summarizes problem characteristics and results obtained for the three problems of each size group. As indicated, the ordinary ($\overline{\text{MFP}}$) relaxations provide very poor information. Relaxation solution values are only 25-50% of optima.

The above (DFP) algorithm was set to stop when either $\nu_P^*/\nu_D^* \leq 102.5\%$ or a 15 minute time limit (CDC Cyber 74) was reached. All problems of less than 1000 arcs stopped before time limit. As indicated, the 1000 arc cases reached solutions provably within 4-8% of optimal in the 15 minutes.

Although this amount of computer time is not insignificant, and results are highly preliminary, values in Table 1 strongly suggests that disaggregated relaxation approaches to fixed charge network problems have great promise. Existing branch-and-bound algorithms for such problems (e.g. Rardin and Unger [9]) are taxed at 100-200 fixed charge arcs because of poor ($\overline{\text{MFP}}$) bounds. With (DFP) it appears 1,000 or more arc problems are within range.

Table 1. Preliminary Computational Results

Problem Size			Estimated % $\nu(\overline{MFP})$	CDC Seconds to Reduce ν_P^*/ν_D^*
Arcs	Nodes	Demands	Forms of $\nu(MFP)$	$\leq$ 102.5% with (DFP)
50	20	5	43.5%	0.8
			23.2%	0.8
			54.6%	5.3
100	36	10	47.3%	7.5
			37.1%	3.8
			36.9%	2.7
200	67	20	36.1%	23.5
			37.0%	19.2
			41.3%	19.6
500	157	50	35.9%	416.5
			40.1%	353.2
			47.6%	237.6
1000	308	100	37.9%	105.5% in 900
			29.3%	107.7% in 900
			41.0%	103.8% in 900

REFERENCES

1. M.L. Balinski, "Fixed Cost Transportation Problems," *NRLQ*, 8, 41-54, (1961).

2. M.L. Balinski, "Integer Programming: Method's, Uses , Computation," *Management Science*, 12, 253-313, (1965).

3. O. Bilde and J. Krarup, "Sharp Lower Bounds and Efficient Algorithms for the Simple Plant Location Problem," *Annals of Discrete Mathematics* 1, (1977). (Based on a 1967 technical report in Danish).

4. P. S. Davis and T. L. Ray, "A Branch-Bound Algorithm for the Capacitated Facilities Location Problem," *NRLQ*, 16, 331-344, (1969).

5. D. Erlenkotter, "A Dual-Based Procedure for Uncapacitated Facility Location," working paper No. 261, Western Management Science Institute, University of California, Los Angeles, (July 1977).

6. Marshall L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, 27, 1-18, (1981).

7. A.M. Geoffrion and G.W. Graves, "Multicommodity Distribution System Design by Benders Decomposition," *Management Science*, 20, 822-844, (1974).

8. T. L. Magnanti and R. T. Wong, "Accelerating Benders Decomposition: Enhancement and Model Selection Criteria," *Operations Research*, 29, 464-484, (1981).

9. R. L. Rardin and V. E. Unger, "Solving Fixed Charge Network Problems with Group Theory Based Penalties," *Naval Research Logistics Quarterly*, 23, 67-84, (1976).