

Slave. On the other hand, to our knowledge, there exists no efficient algorithm which can be used as the Master on this framework. (Lau *et al.* 00) proposed a simple algorithm based on tabu search, which does not seem to be effective, since it does not make use of the underlying network structure.

In this paper, we devise an elegant and rigorous local search approach to solve the Master problem. We demonstrate how to perform efficient and effective neighborhood search in a complex local search space through the use of network sensitivity analysis. In essence, the choice of the next move is determined by solving an instance of minimum cost flow. Even though it is known that the minimum cost flow problem is polynomial-time solvable, since our algorithms make extensive use of minimum cost flow, we need to solve the minimum cost flow problem as efficiently as possible. Instead of solving each instance from scratch, by the nature of local search, it turns out that we can apply the theory of sensitivity analysis to obtain a solution by restarting from the previous solution (basis). For this purpose, we adopt the network simplex method whose strength lies in the ability to perform sensitivity analysis quickly.

As an example applying network flows to solve NP-hard problems, Xu and Kelly proposed a heuristic algorithm for VRP (Xu and Kelly 96). Their approach, an extension of ejection chains model (Glover 96), performs neighborhood searches approximately and efficiently by solving network flow problems. While there are significant differences between their approach and our model, both make use of a minimum cost flow problem, which can be solved quickly, as the background engine.

Preliminaries

Problem Definition

In this paper, we consider a basic version of IRPTW of a single item supplied by a single supplier. Our approach can be easily extended to the more general case involving multiple items and multiple suppliers. An IRPTW instance consists of the following inputs:

- C : set of retailers (customers);
- T : consecutive days in the planning period $\{1, 2, \dots, n\}$;
- d_{it} : demand of retailer i on day t ;
- q_V : vehicle capacity;
- q_W : warehouse storage capacity;
- q_i : storage capacity of retailer i ;
- W_{it} : time window of retailer i on day t ;
- c_i^h : holding cost per unit item per day at retailer i ;
- c_i^b : backlog cost per unit item per day at retailer i ;

The outputs are as follows:

- (1) the distribution plan, which is denoted by x_{it} : integral flow amount from the warehouse to retailer i on day t ;
- (2) the set of daily transportation routes Φ , which carry the flow amounts in (1) from the warehouse to the retailers such that the sum of the following linear costs is minimized:

- a) holding costs and backlog costs
- b) transportation cost from the warehouse to the retailers (T_{ik}).

We use indices i and t for retailers and days respectively.

The distribution plan must obey the demands and storage capacity constraints, and the transportation routes must obey the standard routing, vehicle capacities and time windows constraints. For notational convenience, we let Φ_t denote the set of routes for day t . Each route is an ordered list of retailers representing the delivery sequence performed by one particular vehicle per day.

IRPTW can be regarded either as a generalization of VRPTW or the dynamic lot-sizing problem. In the former case, IRPTW can be seen as a multi-day version of VRPTW, where the daily plans are related via holding and backlog quantities. In the latter case, IRPTW is a *constrained* version of the dynamic lot-sizing problem, whose set-up cost coefficients are defined by delivery (i.e. route) costs. Let this problem be denoted DLP.

Minimum Cost Flow Model for DLP

In this section, we explain our minimum cost flow model solving DLP.

Recall that VRPTW solver outputs a set of routes Φ to DLP. The following variables can be derived directly from Φ : h_{ir} : 1 if retailer i is served by route $r \in \Phi_t$ on some day t , and 0 otherwise; c_r^t : cost of route $r \in \Phi_t$ on some day t , defined as the sum of transportation costs T_{ik} over all adjacent retailers i and k on route r .

The following intermediate variables are used:

z_{it} : integral amount held in retailer i on day t ; b_{it} : integral amount of backlog for retailer i on day t ; y_r : boolean variable, whether route $r \in \Phi_t$ on some day t is used;

The underlying model is a 4-layer network for warehouse V_1 , days V_2 , routes V_3 and retailers (customers) V_4 respectively. Each customer vertex is replicated n times for the n -day planning period. The day, route and customer vertices in the network are denoted by v^d , v^r and v^c respectively. Edges between vertices of different layers represent the flow amounts (warehouse to retailers), while edges between adjacent replicated vertices represent either inventory carrying over to the next day (E_4), or backlog from the previous day (E_5). Formally, the network model is defined as follows (see Fig. 1):

$$V = \bigcup_{i=1}^4 V_i \text{ and } E = \bigcup_{i=1}^5 E_i,$$

where

$$\begin{aligned} V_1 &= \{s\}, \\ V_2 &= \{v_t^d \mid t \in \{1, 2, \dots, n\}\}, \\ V_3 &= \{v_r^r \mid r \in \Phi_t \text{ and } t \in \{1, 2, \dots, n\}\}, \\ V_4 &= \{v_{it}^c \mid i \in C \text{ and } t \in \{1, 2, \dots, n\}\}, \\ E_1 &= \{(s, v_t^d) \mid t \in \{1, 2, \dots, n\}\}, \\ E_2 &= \{(v_t^d, v_r^r) \mid r \in \Phi_t \text{ and } t \in \{1, 2, \dots, n\}\}, \\ E_3 &= \{(v_r^r, v_{it}^c) \mid h_{ir} = 1, r \in \Phi_t \text{ and } t \in \{1, 2, \dots, n\}\}, \end{aligned}$$

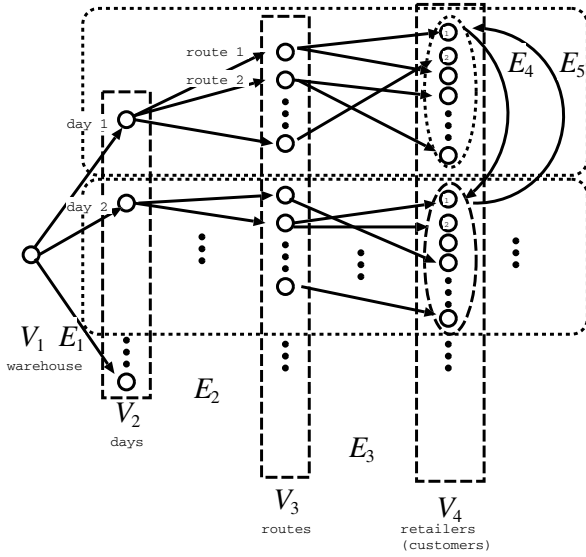


Figure 1: network flow model

$$E_4 = \{(v_{it}^c, v_{i,t+1}^c) \mid i \in C \text{ and } t \in \{1, 2, \dots, n-1\}\},$$

$$E_5 = \{(v_{it}^c, v_{i,t-1}^c) \mid i \in C \text{ and } t \in \{2, 3, \dots, n\}\}.$$

A demand of each vertex is defined as

$$d(v) = \begin{cases} -\sum_i \sum_t d_{it} & v \in V_1, \\ d_{it} & v_{it}^c \in V_4, \\ 0 & \text{otherwise.} \end{cases}$$

A lower bound of each edge is 0 and an upper bound (capacity) u_e is defined as

$$u_e = \begin{cases} q_W & e \in E_1 \\ M_{\text{big}} y_r & e \in E_2 \\ M_{\text{big}} & e \in E_3 \\ q_i & e = (v_{it}^c, v_{i*}^c) \in E_4 \cup E_5 \end{cases} \quad (1)$$

where M_{big} represents a very large value. The cost of each edge is defined as follows:

$$\text{cost}(e) = \begin{cases} c_r^r y_r & e = (v_t^d, v_r^r) \in E_2, \\ c_i^h z_{it} & e = (v_{it}^c, v_{i,t+1}^c) \in E_4, \\ c_i^b b_{it} & e = (v_{it}^c, v_{i,t-1}^c) \in E_5, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Hence, total costs of this network is

$$\sum_t \sum_i (c_i^h z_{it} + c_i^b b_{it}) + \sum_r y_r c_r^r,$$

and this flow satisfies quantity constraints obviously; i.e., this DLP is formulated as one kind of minimum cost flow problems.

Although this problem looks like a simple minimum cost flow problem, it is actually NP-hard since y_r 's are 0-1 variables. On the other hand, for fixed y_r 's, this problem becomes easy, i.e., polynomial solvable. Therefore, we can rewrite this problem as follows:

$$\begin{aligned} \min_y \quad & \sum_{r \in \Phi_t} c_r^r y_r + \text{MCF}(y) \\ \text{subject to} \quad & y_r \in \{0, 1\}, \\ & r \in \Phi_t, t = 1, 2, \dots, n, \end{aligned} \quad (3)$$

where $\text{MCF}(y)$ is the optimal value of the above minimum cost flow network problem for a fixed y_r .

Network Simplex Method

To obtain $\text{MCF}(y)$ in (3), we use network simplex method originally proposed in (Dantzig 51) (see also (Ahuja *et al.* 93) for implementation). Since network simplex method is analogous to simplex method for linear programming problem, it is practically fast though it is not a polynomial time algorithm¹.

Another reason why we use network simplex method is that it is easy to implement sensitivity analysis. Since our heuristics algorithm to solve DLP is based on local search that uses the value of $\text{MCF}(y)$ as a guiding measure, we need to solve many minimum cost flow problem instances. Since these minimum cost flow problems usually have similar networks, we can solve new minimum cost flow problem by making use of solution of the *previous* minimum cost flow problem as the basis - hence, sensitivity analysis.

Now, we explain a basic idea of the network simplex method (for short NSM). (For details, see Chap. 11 of (Ahuja *et al.* 93), on which these explanations are based.)

NSM maintains a *spanning tree solution*, which partitions the edge set of network into three disjoint subsets: (1) T , the edges in the spanning tree; (2) L , the non-tree edges whose flow is restricted to value 0; and (3) U , the non-tree edges whose flow is restricted in value to the edges' flow capacities. We refer to (T, L, U) as a *spanning tree structure*.

Now we define the *reduced cost* $c_{uv}^\pi = c_{uv} - \pi(u) + \pi(v)$, where c_{uv} represents a cost of edge (u, v) , and $\pi(u)$ represents a potential on vertex u . For a spanning tree structure (T, L, U) , we have optimal conditions

$$c_{uv}^\pi = 0 \quad \text{for all } (u, v) \in T, \quad (4)$$

$$c_{uv}^\pi \geq 0 \quad \text{for all } (u, v) \in L, \quad (5)$$

$$c_{uv}^\pi \leq 0 \quad \text{for all } (u, v) \in U. \quad (6)$$

For these conditions, the spanning tree solution in which all edges satisfy them is optimal. The reduced cost c_{uv}^π means the cost per unit flow intuitively. For example, if we were to add flow of x units on edge (u, v) , then the total cost of this network increases by $(c_{uv} - \pi(u) + \pi(v))x$ units.

NSM proceeds as follows: It maintains a feasible spanning tree structure such that all tree edges satisfy condition (4) at each iteration and successively transforms it into an improved spanning tree structure until all non-tree also satisfy condition (5) and (6).

Algorithm NETWORK SIMPLEX METHOD:

Step 1: Find an initial feasible tree structure (T, L, U) .

Step 2: If some non-tree edge violates the optimality condition (5) or (6), find an improved spanning tree structure and go to Step 2. Otherwise halt.

Next, we explain sensitivity analysis on edge costs. Consider that the cost of an edge (u, v) increases by λ . (Reducing λ also can be done similarly.) Let (T^*, L^*, U^*) denote

¹Although there exist polynomial time algorithms to solve minimum cost flow problems (Ahuja *et al.* 93), network simplex method is still practical as simplex method for linear programming.

the spanning tree structure of the optimal solution and π^* denote the corresponding vertex potential. The analysis would be different when edge (u, v) is a tree or a non-tree edge.

Case 1. edge (u, v) is a non-tree edge.

Adding the cost of edge (u, v) does not affect the vertex potentials of the current spanning tree structure, since edges in T^* still satisfy optimal condition (4). The modified reduced cost of edge (u, v) is $c_{uv}^{\pi^*} + \lambda$. If its cost satisfies condition (5) or (6), (T^*, L^*, U^*) remains optimal. Otherwise, we reoptimize the solution using NSM with (T^*, L^*, U^*) as the initial spanning tree structure.

Case 2. edge (u, v) is a tree edge.

We have to change some vertices' potentials in order to maintain the current spanning tree structure (T^*, L^*, U^*) , satisfying optimal condition (4). If edge (u, v) is an upward-pointing edge on the current spanning tree, potentials of all the vertices in subtree whose root is u increase by λ , and if (u, v) is a downward-pointing edge, potentials of all the vertices in subtree whose root is v decrease by λ . If all non-tree edges still satisfy the optimality condition, the current spanning tree structure remains optimal; otherwise, we reoptimize the solution using NSM.

Local Search Algorithms for DLP

In this section, we present our local search algorithms for solving DLP. In general, local search starts from an initial feasible solution and repeats replacing it with a better solution in its *neighborhood* until no better solution is found in its neighborhood. Roughly speaking, a neighborhood is a set of solutions obtainable from the current solution by a slight perturbation.

In our problem, the solution is defined as an assignment of $y = (y_1, y_2, \dots, y_k)$, and the cost, $\text{cost}(y)$, of solution y is defined as the value of equation (3). In our algorithm, we use the following neighborhoods:

Delete Neighborhood $N^D(y)$ is defined as $N^D(y) = \{y' \mid y' = y - e^{(j)}, j \in ON(y)\}$, where $ON(y) = \{j \mid y_j = 1\}$ and $e^{(j)}$ is j -th unit vector. This neighborhood represents transition that route j is deleted from the set of routes used in current solution.

Add Neighborhood $N^A(y)$ is defined as $N^A(y) = \{y' \mid y' = y + e^{(l)}, l \in OFF(y)\}$, where $OFF(y) = \{l \mid y_l = 0\}$ and $e^{(l)}$ is l -th unit vector. This neighborhood represents transition that route l is added into the set of routes used in current solution.

Swap Neighborhood $N^S(y)$ is defined as $N^S(y) = \{y' \mid y' = y - e^{(j)} + e^{(l)}, j \in ON(y) \text{ and } l \in OFF(y)\}$ where ON, OFF and e^* are as defined in Delete and Add neighborhood.

We apply moves to solutions in $N^D(y)$, $N^A(y)$ and $N^S(y)$, *delete operation*, *add operation* and *swap operation*, respectively. Using these neighborhoods and operations, we propose the following LS (local search) algorithm:

Algorithm LS:

Step 1: (initialize) Find an initial solution σ_{init} (see below), and set $\text{best} := \text{cost}(\sigma_{\text{init}})$.

Step 2: (improve by LS) Improve σ by neighborhood search. set $\text{improve} := \text{false}$.

(2-a): If there exist $y' \in N^A(y)$ such that $\text{cost}(y') < \text{cost}(y)$, set $y := y'$, $\text{improve} := \text{true}$, and return to (2-a). Otherwise go to (2-b).

(2-b): Do likewise for neighborhood N^D .

(2-c): Do likewise for neighborhood N^S

Step 3: (halt or repeat) If $\text{improve} = \text{false}$, output y and stop; otherwise return to Step 2.

In Step 1, we use a simple heuristics algorithm *initial deletion* in order to find an initial solution y .

Algorithm INITIAL DELETION:

Step 1: Set $j := 0$, $y = (1, 1, 1, \dots, 1)$, and an order $R_o = \{r_1 \preceq r_2 \preceq \dots \preceq r_k\}$ of route $r \in \Phi_t$ for all t , according to the value c_r^r/w_r .

Step 2: Set $j := j + 1$. If $j = k + 1$, output y and halt. Otherwise go to Step 3.

Step 3: If $\text{cost}(y') < \text{cost}(y)$ where $y' = y - e^{(r_j)}$, set $y := y'$. Return to Step 2.

Since the value c_r^r/w_r in Step 1 can be considered the redundancy of route r , this algorithm is essentially a greedy algorithm.

Local search method has a disadvantage that it is impossible to escape from local optima; i.e., y does not have better solution in its neighborhood $N(y)$, but is not a global optimal. Therefore, we implement two meta-heuristics algorithms ILS and TS.

Iterated local search

Iterated local search (ILS) (Gu 92; Johnson 90) generates initial solutions of local search, by slightly perturbing a solution y_{seed} , which is a good (not necessarily the best) solution found during the search. This diversity of initial solutions lead us to escape from local optima. Our ILS works as follows:

Algorithm ILS

Step 0: (initialize) Generate an initial solution y_{seed} by Initial Deletion, and set $\text{best} := \text{cost}(y_{\text{seed}})$.

Step 1: (generate an initial solution) Generate a solution y by slightly perturbing y_{seed} .

Step 2: (improve by LS) Improve y by LS, i.e., set $y := LS(N, y)$.

Step 3: (update the best and seed solutions) If $\text{cost}(y) < \text{best}$, set $\text{best} := \text{cost}(y)$ and $y^* := y$. If some accepting criterion is satisfied, set $y_{\text{seed}} := y$.

Step 4: (halt or random restart) If some stopping criterion (computational time or the number of iterations exceeded) is satisfied, output y^* and stop; otherwise return to Step 1.

In Step 1, the new solution y is generated by l -Jump Operation.

l -Jump Operation First, choose a set L of components from $\{1, 2, \dots, k\}$ randomly, where $|L| = l$. For input y , flip values of components in L , i.e., for $i \in L$,

$$y_i := \begin{cases} 1 & y_i = 0, \\ 0 & y_i = 1. \end{cases}$$

Tabu search

Tabu search (Glover and Laguna 97) tries to enhance LS by using the memory of the previous search. The best solution in $N(y) \setminus (\{y\} \cup T)$ is chosen as the next solution, where the set T , called *tabu list*, is a set of solutions which includes those solutions most recently visited. Usually the size of tabu list $|T|$ is less than t_{tenure} , called *tabu tenure*. In tabu search, a move to a new solution is always executed even if the current solution is locally optimal. This causes cycling of solutions in general. Introducing T enables the algorithm to avoid cycling in a short period.

In our tabu search (TS), we keep tabu list as 1-array of size k , associated with routes, and put τ_j . τ_j is defined as the iteration number when an operation is done about route j most recently. For the current iteration number t_{it} , if $t_{\text{it}} - \tau_j < t_{\text{tenure}}$, all operations about j are forbidden, i.e., tabu. TS works as follows:

Algorithm TS

Step 1: (initialize) Generate a solution y by Initial Deletion, set $y^* := y$ and $T := \emptyset$.

Step 2: (decide a move) *improve* := **false**. Find the best solution y' in $N(y) \setminus (\{y\} \cup T)$ for (2-a),(2-b) and (2-c), and set $y := y'$.

(2-a): Find the best solution $y' \in N^A(y)$ and set $y := y'$. If $\text{cost}(y') < \text{cost}(y)$, *improve* := **true**.

(2-b): Do likewise for neighborhood N^D

(2-c): Do likewise for neighborhood N^S

Step 3: (update the best cost) If $\text{cost}(y) < \text{cost}(y^*)$, set $y^* := y$ and $t_{\text{tenure}} := 0.9 \times t_{\text{tenure}}$.

Step 4: (update t_{tenure}) If *improve* = **false**, $t_{\text{tenure}} := 1.1 \times t_{\text{tenure}}$.

Step 5: (halt or further search) If some stopping criterion is satisfied, output y^* and stop; otherwise return to Step 2.

In general, more features such as *long term memory* and *aspiration level* are introduced in the framework of tabu search. For this paper, we adopt the above simple tabu search.

Sensitivity Analysis Induced by Local Search

In local search, we have to solve minimum cost flow problems many times, and it is quite time-consuming in general. However, since the computation is neighborhood search, these minimum cost flow problems have almost same structures (networks) except few edges coefficients. For example, a delete operation $y := y - e^{(r)}$ is considered changing 0 into M_{big} on the cost of the corresponding edge in the network. (We adopt sensitivity analysis about not the capacities coefficient but the cost one, since the cost one is a little simpler to implement and we can obtain the same effect.)

Changes about cost coefficient are (1) adding M_{big} (delete operation), (2) reducing M_{big} (add operation). A swap operation is performed as a combination of (1) and (2). Both of them can be performed by putting $\lambda = M_{\text{big}}$ (or $-M_{\text{big}}$) in the Network Simplex Method described above.

Numerical Experiments

In this section, we report experimental results. To our knowledge, no benchmark test data available in the literature matches our problem exactly. Hence, our experimental results are based on the test data generated from Solomon's VRPTW benchmark problems (Solomon 87), as follows.

We set the planning period $n = 10$. For each day in the planning period, we set the vehicle capacity, locations and time-windows of the retailers and warehouse (depot) to be equal to those specified in the Solomon instance. We create demand d_{it} of retailer i for day t from the demand d_i of Solomon instance, by partitioning $n \times d_i$ into n parts, i.e., $d_{i1}, d_{i2}, \dots, d_{in}$ randomly such that d_{it} is within the range $[0.5 * d_i, 1.5 * d_i]$ for $t = 1, 2, \dots, n$. The capacities of retailers and warehouse are set as vehicle capacity and M_{big} respectively. As for cost coefficients, holding cost of retailer (c_i^h) and backlog cost (c_i^b) are set to be 1 and 2 for all i respectively. The transportation cost of each route is set to be 10 times its total distance.

The DLPsolver is implemented by our algorithms described above. The VRPTW solver is based on a standard two-phase heuristics. Both TS and ILS adopt the halting condition that CPU run time reaches 180 seconds.

We run our program on a Pentium 666MHz PC and the results are given as follows. In this table, we plot the different test instances against: (1)“VRPTW”: initial objective value by using only VRPTW solver, (2)“ILS+VRP”: final objective value where DLP is based on ILS and (3)“TS+VRP”: final objective value where DLP is based on TS.

Data	VRPTW	ILS+VRP	TS+VRP
c201	178650	113263	112821
c202	192818	117483	124312
c203	200615	131920	122055
c204	216447	136384	142300
c205	175378	116147	109248
c206	177331	123978	127876
c207	177447	122204	117735
c208	175268	124110	125667
r201	304779	111330	116893
r202	291492	116982	114717
r203	247122	110215	115070
r204	227381	114118	114118
r205	284759	122333	123009
r206	260760	120928	123251
r207	223527	115438	115438
r208	228033	120011	117255
r209	249036	116840	120725

Table 1: Computational results of our algorithms

From Table 1, we observe that the improvement in the objective value of the final solution (2) over the VRPTW solution (1) is at least 29.1% in c208 and at best 39.0% in c202 (among C-instances). As for (3), the worst improvement is 27.9% in c206 and the best is 39.2% in c203. In case of R-instances, improvement ration is from 48.4% to 61.7% on TS, and from 47.4% to 63.5%.

Our approach seems useful for these instance. However, these improvement ratio seems to depend on the difficulty

of the instances, as we can see the difference of the results between C- and R-instances. We also believe the difficulty of our IRPTW instances are influenced by cost coefficients.

Next, we show the effectiveness of sensitivity analysis. Recall that our algorithms need to invoke minimum cost flow many times. We implemented our algorithms in two ways: (a) solving each minimum cost flow problem from scratch; and (b) solving each minimum cost flow problem from the previous problem's solution (i.e., apply sensitivity analysis).

Table 2 shows the comparison between ways (a) and (b) for R2 Solomon instances. Each column shows the total number of pivots on NSM and total execution time. The third column shows the ratios of (a) to (b) rounded down to the nearest integer.

	(a) Scratch		(b) Sensitivity		Ratio	
	Pivots	Time	Pivots	Time	Pivots	Time
r201	19741024	113.83	423760	2.77	46	41
r202	11484659	65.9	300424	2.13	38	30
r203	4731487	25.86	280710	1.8	16	14
r204	3349778	18.51	264430	1.5	12	12
r205	10130292	57.57	346624	2.37	29	24
r206	3860581	21.83	281394	1.91	13	11
r207	5995470	34.94	230282	1.39	26	25
r208	5351521	31.49	281394	1.7	19	18
r209	8905561	51.43	326324	1.96	27	26

Table 2: Effectiveness of sensitivity analysis

From Table 2, the following observations can be made:

1. In terms of number of pivots, the ratio is at least 12.67, at most 46.59, and 25.514 on average. As for run time, the ratio is at least 12.34 times, at most 41.09 times, and 22.71 on average.
2. The pivot ratios are always slightly larger than time ratios.

Observation 1 shows sensitivity analysis on NSM is effective indeed. One reason for Observation 2 is as follows: In Step 1. of NSM, we add some artificial edges to the original network (see (Ahuja *et al.* 93)). These artificial edges make a new improved spanning tree structure simple, and make it easy to find a new improved spanning tree structure in the early iterations (pivots) on NSM. By contrast, sensitivity analysis use a network of the previous solution, which usually has more complicated form. Therefore, we suppose that one pivot in sensitivity analysis takes longer time than one pivot in original NSM.

Conclusion

In this paper, we proposed algorithms for solving IRPTW, based on the framework introduced in (Lau *et al.* 00). These algorithms are rigorous in the sense that they make use of the underlying network structure as a guidance measure for search. As a technical contribution, we demonstrate how to perform efficient and effective neighborhood searches on local search through the use of network sensitivity analysis. We believe that our approach can be adapted quickly to solve other complex network optimization problems

such as those arising in logistics and telecommunications applications.

References

- R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows : Theory, Algorithms and Applications*, (Prentice-Hall, 1993).
- A. Campbell, L. Clarke, A. J. Kleywegt, and M. W. P. Savelsbergh, The Inventory Routing Problem, in: T. G. Grainic and G. Laporte, eds., *Fleet Management and Logistics*, (Kluwer Academic Pub., 1998), 95–113.
- M. W. Carter, J. M. Farvolden, G. Laporte, J. Xu, Solving an Integrated Logistics Problem Arising in Grocery Distribution, *INFOR*, **34:4** (1996), 290–306.
- Y. Caseau and T. Kokeny, An Inventory Management Problem, *The Constraints Journal*, **3**, (1998), 363–373.
- L. M. Chan, A. Fedegruen and D. Simchi-Levi, Probabilistic Analysis and Practical Algorithms for Inventory-Routing Models, *Oper. Res.*, **46:1** (1998) 96–106.
- G. B. Dantzig, Application of the simplex method to a transportation problem, *Activity Analysis and Production and Allocation*, edited by T. C. Koopmans, (Wiley, 1951).
- M. Florian, J. K. Lenstra, and A. H. G. Rinnooy Kan, Deterministic Production Planning: Algorithm and Complexity, *Management Sc.*, **26:7** (1980), 669–679.
- A. Fedegruen and P. Zipkin, An Efficient Algorithm for Computing Optimal (s,S) Policies, *Operations Research*, **22** (1984), 1268–1285.
- F. Glover, Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems *Discrete Applied Mathematics*, **65**, (1996) 223–253.
- F. Glover and M. Laguna, *Tabu Search*, (Kluwer Academic Publishers, 1997).
- S. C. Graves, A. H. G. Rinnooy Kan and P. H. Zipkin eds., *Handbook in Operations Research and Management Science Vol 4: Logistics of Production and Inventory*, (North-Holland, 1993).
- J. Gu, Efficient Local Search for Very Large-Scale Satisfiability Problem, *SIGART Bulletin*, **3**, (1992), 8–12.
- D. S. Johnson, Local Optimization and the Traveling Salesman Problem, *Proceedings of the 17th Colloquium on Automata, Languages and Programming* (1990) 446–461.
- H. C. Lau, A. Lim and Q. Z. Liu, Solving a Supply Chain Optimization Problem Collaboratively, *Proc. 17th National Conf. on Artificial Intelligence (AAAI)*, (2000) 780–785.
- M. W. P. Savelsbergh, Local Search for Routing Problems with Time Windows, *Annals of Operations Research*, **4**, (1986), 285–305.
- M. M. Solomon, Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints, *Operations Research*, **35**, 1987, 254–265.
- J. Xu and J. P. Kelly, A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, *Transportation Science*, **30:4**, (1996) 379–393.