Scuola universitaria professionale
della Svizzera italiana
SUPSI
Dipartimento tecnologie innovative

MSE | MASTER OF SCIENCE IN ENGINEERING

Intelligent Systems and Networking
Information and Communication Technologies

Column Generation for a Rich VRP

VRP with simultaneous distribution, collection and pickup-and-delivery

Lorenzo Ruinelli

February 2011

Supervisors
Prof. Luca Maria Gambardella
Dr. Matteo Salani

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Lorenzo Ruinelli
Manno, February 2011

Abstract

We present an optimization algorithm developed for a real-world distribution logistic company. The algorithm computes a daily plan for a heterogeneous fleet of vehicles that depart from one depot and must visit a set of customers for delivery operations. In this rich vehicle-routing problem (VRP), we consider: (i) heterogeneous fleet of vehicles (ii) departure from different locations (iii) multiple time windows associated with locations (iv) incompatibility constraints between goods, vehicles and locations (v) maximum route length and duration (vi) the possibility of open routes that do not terminate at the depot (vii) the cost of each vehicle route is computed through a system of fees (viii) the customers demands are made of deliveries, collections and pickup-and-delivery (ix) driving and working time rules.

We devise an algorithm which combines three components to produce heuristically VRP solutions with a measures of their quality. The components are an Ant Colony System, a Column Generation algorithm and a general purpose mixed-integer linear problem solver. Column Generation component has been designed and implemented by us, while the others two are used as black-boxes. We describe the implementation details of the Colung Generation component and we devise an incremental heuristic.

We present computational results on 14 real instances obtained from the distribution logistic company. Moreover, we report an analysis of 2 hand-made solutions provided by the company planners.We are able to improve the solution quality in much less computing time. Additionally, we respect all the operational constraint, while hand-made plans often violate them.

Acknowledgements

Contents

Figures

Chapter 1

Introduction

Vehicle Routing Problem (VRP) calls the determination of the optimal set of routes to be performed by a fleet of vehicles to serve a given set of customer.

The problem was introduced by Dantzig and Ramser (1959) as a generalization of the Travel Salesman Problem (TSP). Solve such a problem is an hard task because it has no known polynomial-time solution that is guaranteed to provide an optimal solution. This quality implies that the problem is difficult to deal with in practice. VRP can be modelled as a Set Covering Problem which is known to be NP-complete (Garey and Johnson, 1979).

VRP are widely studied in Operation Research. Since its introduction, hundreds of models and algorithms (mostly based on Mathematical Programming and Combinatorial Optimization) were proposed for the optimal and approximate solution of the different version of the VRP.

An efficient solution of this optimization problem is highly relevant for real-world logistic companies (Golden et al., 2008) (Ceselli et al., 2009). The last decades has seen an increasing utilization of optimization package for the effective management of the provision of goods and services in distribution system. The use of computerized optimization for the distribution process planning produce substantial savings (generally from 5% to 20%) in transportation costs. This result appears even more significant because the transportation cost involves all stages of the production and represent a relevant component (generally from 10 % to 20%) of the final cost of good (Toth and Vigo, 2002).

The main methods for the solution of hard combinatorial optimization problems can be exact or heuristic. Exact methods produce an optimal solution with guarantee via an enumeration tree (Wolsey, 1998). These methods are based on efficient computation of valid primal and dual bounds. On the other hand, heuristic methods do not guarantee optimal solution but are in general much faster than exact methods. Popular metaheuristics for combinatorial problems include ant colony system (Dorigo and Gambardella, 1997) and genetic algo-

rithms (Holland, 1975). They iteratively try to improve a candidate solution with regard to a given measure of quality (e.g. the total km of the set of routes).

When we solve a VRP instance we deal with a search-space of candidate solutions which grows exponentially as the size of the problem increases. Formulations with exponential number of columns can not be threated explicitly. For this reason, exact methods for VRP traditionally suffer because of the need to enumerate the whole search space (e.g. as variables within a linear problem) which makes an exhaustive search for the optimal solution infeasible. Column Generation deals only implicitly with the variables of the problem. Dantzig and Wolfe (1960) pioneered this fundamental idea, developing a strategy to extend a linear program columnwise as needed in the solution process . Column Generation is nowadays a prominent method to cope with a huge number of variables. It has been used as the main component of many exact algorithms for a large class of integer programs.

Recent contributions to the exact solution of realistic versions of the VRP are mainly based on Branch&Price&Cut. In its basic version, the VRP is solvable to optimality for instances with up to 200 customers (Baldacci et al., 2008). Problems with time windows are considered in Desaulniers (2010). Advances in real-world feature modeling consider splittable loads in which each customer can be served by more than one vehicle as in the problem addressed by this thesis (Archetti et al., 2006; Sharda et al., 2008; Nowak et al., 2009; Desaulniers, 2010). More recently, load-dependent service time has been modeled by Salani and Vacca (2009) who address a discrete split version of the VRP with time windows using a branch-and-price algorithm based on column generation and present computational results on instances based on Solomon's data set. Whereas, Ceselli et al. (2009) proposes a branch and price algorithm to solve a real-world VRP similar to the one addressed in this thesis and presents computational results on real instances obtained from a provider of software-planning tools for distribution logistics companies. Rizzoli et al. (2007) discuss the application of heuristic methods based on Ant Colony Optimization to a number of real-world problems (VRP with time windows, VRP with pickup and delivery, time dependent VRP where the travel times depend on the time of the day and on-line VRP where customers' orders arrive during the delivery process).

In this thesis we propose a column generation based heuristic procedure to solve a real-world VRP. Our algorithm produces daily plans for a distribution logistic company exploiting the Dantzig-Wolfe decomposition of the flow formulation of the original problem into a master problem and a pricing subproblem (we refer the reader to Lubbecke and Desrosiers (2005) which provides a comprehensive review of contributions related with the integer programming column generation process). The master problem is a set-covering problem with binary

variables. In our implementation, we consider specific issues of column generation: initialization and column management strategies (see Vanderbeck (2005)). The pricing subproblem is a constrained elementary shortest-path problem that is iteratively solved to produce new promising columns. We solve the pricing problem using the dynamic programming technique pionereed by Desrosiers et al. (1981) and improved in the last decades by Righini and Salani (2006) using a bounded bi-directional search and decremental state space Righini and Salani (2008). Finally, we considered some preprocessing techniques for the elementary shortest-path problem proposed by Dumitrescu and Boland (2003).

In chapter 2, we describe the problem. In chapter 3, we report the mathematical model used to solve the problem using column generation. In chapter 4, we describe the used algorithms and we devised an incremental heuristic based on column generation which starts from a reduced network and increases it iteratively. In chapter 5, we solved 14 real-world instances using various settings of our algorithm and we provided conclusive remarks. Chapter 6 contains an analisys of 2 hand-made solutions provided by the company planner which are compared with the results of our algorithm. In appendix A, we report the summary of the scientific articles used in the thesis. In appendix B, we report an example of a relaxed master problem and we shows how it changes while column generation itarates. Finally, appendix C describes the architecture of the implemented software.

Chapter 2

Problem description

The algorithm solves the problem of a real-world distribution logistic company which has to compute the daily plan of his fleet of vehicle. That company operates in the field of logistics under controlled temperature and their customers are distributed in Italy. Ours relations with the distribution company were mediated by AntOptima which is a provider of software planning tools. AntOptima studied as first the problem and gave us all its characteristics and constraints.

The problem is a rich vehicle-routing problem (VRP) considering: (i) heterogeneous fleet of vehicles (ii) departure from different locations (iii) multiple time windows associated with locations (iv) incompatibility constraints between goods, vehicles and locations (v) maximum route length and duration (vi) the possibility of open routes that do not terminate at the depot (vii) the cost of each vehicle route is computed through a system of fees (viii) the customers demands are made of deliveries, collections and pickup-and-delivery (ix) driving and working time rules.

2.1 Problem characteristics

Locations The distribution network of our problem consists of two types of locations: customer sites and one depot. The position of each location is known and defined by a five-level hierarchical code: nation, zone (primary and secondaries), region, district, and the node (e.g. customer site IPER CARNI SRL is a geographical node of latitude 41.865879 and longitude 12.603657, it lies in the district of Rome. Rome is in the region of Lazio and Lazio is in Italy. Rome has primary zone 10 and secondaries zones 8, 11).

The distance between each pair of locations, in terms of both kilometers and traveling time, is known as well and depends on vehicle type. Due to different road types and average traffic conditions, distance in time and in space may not

be proportional.

For both location types (customers and depots), we are given a set of time windows in which loading and unloading operations can take place.

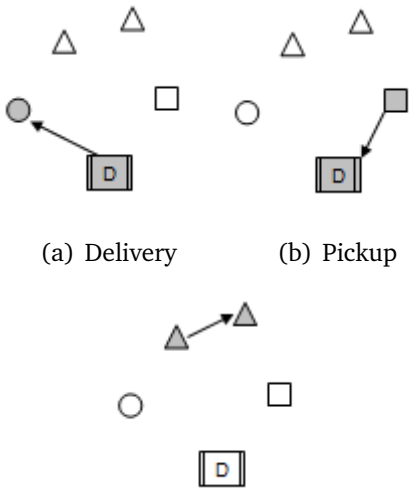
Each customer's site has a delivery dependent load/unload service time functions (e.g., for a customer site with a fixed unload time of 10 min. and a variable unload time of 5 min./pallet, a delivery of 11 pallets would last $10+5\times11=65$ min.). The service time is computed in pre-processing since orders can not be splitted up.

Service types The logistic company provides 3 types of services:

1. Delivery: an item is collected from the depot and delivered to another location.

2. Pickup: an item is collected from a customer and delivered to the depot.

3. Pu-and-del: an item is collected from a customer and delivered to another customer. We defined the collection part as $pu-and-del_{in}$ and the delivery part as $pu-and-del_{out}$.



(a) Delivery (b) Pickup

(c) Pu-and-del

Figure 2.1: Services types

	constraints				
	stop		type of services		
utiliz.	departure	arrival	delivery	pickup	pu-and-del
AR	depot	depot	>= 1	free	free
A	depot	free	>= 1	free	free
R	not depot	free	0	free	free

Table 2.1: Utilization-types of vehicles

Vehicles Operations must be done using a given vehicle fleet. The fleet is made of a set of non homogeneous vehicles. A vehicle has a daily duty which is composed of one tour.

Each vehicle type is described by several data: the availability of an hydraulic lift (sponda idraulica), the availability of a transpallet, the maximum weight, the maximum volume, the maximum number of pallets that the vehicle can carry.

Each vehicle has an utilization-type which defines some constraints on its tours. The utilization-type affects the departure stop, the arrival stop and the services a vehicle can do within its tours. The existing utilization-types are AR, A and R (see Table 2.1). Vehicles of utilization-type AR must depart and arrive to the depot and have to delivery at least one item. Vehicles of utilization-type A must depart from the depot and has to delivery at least one item. Vehicles of utilization-type R can not deliver items since they do not depart from the depot, they can only do collection and pu-and-del.

Tour A tour is composed of one or more stops. By "stop" we mean a visit to a customer to perform a service (in the case of a pu-and-del, the stops would be 2, see Figure 2.1(c)).

The stops within a tour must respect some precedence stops constraints, in fact deliveries must be done before pickups, pickup-and-delivery can be done in any order (before, after and between delivery; before, after and between pickups) but the destination must immediately follow the origin. For instance, the tour depot-delivery-delivery-pickup-depot is valid, conversely the tour depot-delivery-pickup-delivery-depot isn't (because it does a pickup before a delivery).

A tour is associated with a region which defines its maximal length (in terms of both time and distance).

2.2 Operational constraints

Time Windows Each location can be visited within a given time windows (single or double). Early arrivals are permitted but service starts when the time window opens.

Primary and secondary zones The locations which a vehicle can visit in the future depend on the primary and secondary zones of the locations visited in the past. Each zone is associated with an upper bound on tour length. The zones visited by a vehicle define the zone of a tour and consequently its maximum length limit.

In order to explain the primary and secondary zone constraint we propose an example using following data structures: $primz$ and $secondz$ are the sets containing the primary and the secondary zones of the locations where the stop is done. Z NEXT STOP is a set containing the intersection of the visited zones and determines the visitable zones in the reminder of the tour. Z TOUR contains the zone id whithin the set of feasible zones with the highest kilometric limit

Listing 2.1 reports the example, which is a tour doing 4 stops. Line 3 reports the first stop, which is done in a location with primary zone 1 and secondary zones 2,3 and 4 the value of Z NEXT STOP is then $primz \cup secz = \{1\} \cup \{2,3,4\} = \{1,2,3,4\}$. For the second stop of the example, locations with $primz \cup secz \cap ZNEXTSTOP \neq \emptyset$ are allowed. $\{2\} \cup \{3,4\} \cap \{1,2,3,4\} = \{2,3,4\} \neq \emptyset$ and $\{2,3,4\}$ is the new value of Z NEXT STOP. We note how 1) we use Z NEXT STOP to check the feasibility of the new stop 2) we update Z NEXT STOP according with the new stop. The value of Z TOUR is used to compute the tour length validation, line 9 reports a Z TOUR with value 3 (we choose 3 instead of 4 because it has the associated biggest limit, in fact MAX KM ZONA3 > MAX KM ZONA4) which has an associated limit of MAX KM ZONA3 = 1200 this value is compared with the total length of the tour to do the validation (line 15).

Listing 2.1: Primary and secondary zones

```
1  SERVICES DONE IN EACH STOP:
2    00 depot
3    01 delivery (primz={1} secz={2,3,4})    Z_NEXT_STOP={1,2,3,4}
4    02 delivery (primz={2} secz={3,4})      Z_NEXT_STOP={2,3,4}
5    03 delivery (primz={3} secz={4})        Z_NEXT_STOP={3,4}
6    04 pickup
7    05 depot
8  VALUES OF THE TOUR:
9    Z_TOUR = MAX(MAX_KM_ZONA3, MAX_KM_ZONA4) = 3
10   LENGTH_KM=1264
11 PARAMETERS:
```

```
12    MAX_KM_ZONA3=1200
13    MAX_KM_ZONA4=999
14  VALIDATION:
15    if 1264 < 1200 then THE CHECK IS OK
16    else THE CHECK IS NOT OK
```

Driving Time and Working Time The driving time (i) and the working time (ii) can not exceed upper bounds defined at regional level. The parameters used are TEMPO DI GUIDA XXX (e.g. TEMPO DI GUIDA EMI=720 means a maximum driving duration of 720 minutes in region Emilia) and
TEMPO DI LAVORO XXX (e.g. TEMPO DI LAVORO LAZ=960 means a maximum working duration of 960 minutes in region Lazio).

The working time is the sum of load times, drive times and unload times (load and unload times into the depot are not considered).

The region used to select the correct parameters is that of the first stop of the tour.

Listing 2.2 is an example of driving and working time computation. It represents a tour with 4 stops, we reports the detail of the times it is made of and the region of the first stop (PIE, which is used to select the correct upper bound). The tour is validated.

Listing 2.2: An example of driving and working time computation

```
SERVICES DONE IN EACH STOP:
  00 depot
  01 delivery (piatt) (PIE)
  02 pickup (0 plt)
  03 delivery (diretta)
  04 pickup (36.3 plt)
  05 depot
TIMES OF TOUR:
  00-01: load  60; wait  93; drive  161; wait   0; unload  45
  01-02: load   0; wait   0; drive    2; wait   0; unload   0
  02-03: load   0; wait   0; drive    0; wait   0; unload  44
  03-04: load   0; wait   0; drive  141; wait  96; unload   0
  04-05: load  46; wait   0; drive   87; wait   0; unload  46
VALUES OF THE TOUR:
  DRIVINGTIME=161+2+141+87=391
  WORKINGTIME=161+45+2+44+141+46+87=526
PARAMETERS:
  TEMPO_DI_GUIDA_PIE=720
  TEMPO_DI_LAVORO_PIE=960
VALIDATION:
  (i) 391<=720 (TRUE)
  (ii) 526 <= 960 (TRUE)
```

```
if (i==TRUE) and (ii==TRUE) then THE CHECK IS OK
else THE CHECK IS NOT OK
```

Maximal distance for vehicle performing R-utilization-type services The distance between the departure location of a vehicle performing R-utilization-type services and its first pickup can not exceed a given parameter. The parameter is DISTANZA MASSIMA MEZZI R and its value is typically 40 km.

Weight and volume The amount of good which can be loaded on a vehicle has to respect some constraints. The amount of pallet of pickups and pickup-and-delivery can not exceed the weighted maximum capacity (in pallet) of the vehicle (i). The weighted amount of pallet (calc_plt) of deliveries can not exceed the maximum capacity (in pallet) of the vehicle (ii) (this depends on regional rules which are not detailed here, the amount of calc_plt is computed by preprocessing). The total kg of the good can not exceed the maximum capacity (in kg) of the vehicle (iii).

Listing 2.3 reports an example of weight and volume rules check.

Listing 2.3: An example of weight and volume computation

```
SERVICES DONE IN EACH STOP:
  nr00 depot
  nr01 delivery  (kg 3902,43) (calc_plt 5.9)
  nr02 delivery (kg 1357,572) (calc_plt 3.06)
  nr03 delivery (kg  1349,519) (calc_plt 4.08)
  nr04 delivery (kg 4319,754) (calc_plt 11.39)
  nr05 pickup (kg 0) (plt 20)
  nr06 pickup (kg 0) (plt 12)
  nr07 depot
VALUES OF THE TOUR:
  VEHICLE_MAX_CAPACITY_PLT = 33
  VEHICLE_MAX_CAPACITY_KG = 26000
  MAX_PLT_PU_AND_RITIRO= 20+12=32
  MAX_LOAD_KG=3902,43+1357,572+1349,519+4319,754=10929,275
  CALC_PLT_DELIVERY=5.9+3.06+4.08+11.39=24,49
PARAMETERS:
  SOVRACCARICO_PRESE=115
VALIDATION:
  (i) 32 <=  33 × 115 / 100 (TRUE)
  (ii) 24.49 <=  33 (TRUE)
  (iii) 10929.275 <=  26000  (TRUE)

  if ((i==TRUE) and
      (ii==TRUE) and
      (iii==TRUE)) then THE CHECK IS OK
  else THE CHECK IS NOT_OK
```

Compatibilities Location-vehicle compatibility can be described in terms of vehicle type exclusion (see 2.4) and vehicle characteristic requirement (see 2.5).

Listing 2.4: An example of vehicle type exclusion

```
LOCATION COOP_X:
  CAN NOT BE VISITED BY VEHICLES OF TYPE [BDP;BIL;FIT;MOP;MOT]
LOCATION IPER_Y:
  CAN NOT BE VISITED BY VEHICLES OF TYPE [BIL]
```

Listing 2.5: An example of vehicle characteristic requirement

```
LOCATION COOP_X REQUIRES SPONDA_IDRAULICA

VEHICLE "COND01" HAS SPONDA_IDRAULICA
   CAN VISIT: COOP_X

VEHICLE "COND02" HASN'T SPONDA_IDRAULICA
   CAN NOT VISIT: COOP_X
```

Maximal number of stops The number of stops can not exceed an upper bound given by the parameter MAX FERMATE VIAGGIO.

Maximal distances between stops

- Delivery-Delivery: the distance between two consecutive deliveries can not exceed a regional upper bound.

  Listing 2.6 reports a tours with two deliveries, it is feasible because the distance between the two stops is 55 km and the maximal distance is 500 km (we use the limit of the region of the first stop).

- Delivery-Pickup, Delivery-Pu-and-del$_{in}$: the check is not done if the origin and destination districts are matched by an entry of parameters PROVINCE SENZA LIMITI PRESA (i). The distance can not exceed a regional upper bound (ii). The check considers also the parameter FATTORE DI RIENTRO (iii).

  Listing 2.7 reports a tour doing deliveries and pickups, the check is done between stop 02 (line 4) and stop 03 (line 5). In lines 17,18 and 19 we report respectively check (i), (ii) and (iii). The tour is valid (line 23).

- Pickup-Pickup, Pu-and-del$_{out}$-Pickup, Pu-and-del$_{out}$-Pu-and-del$_{in}$: the distance can not exceed a regional upper bound (i). The check considers also the parameter FATTORE RIENTRO PRESE SUCCESSIVE (ii).

Listing 2.8 reports a tour doing deliveries and pickups, the check is done between stop 03 (line 5) and stop 04 (line 6). In lines 16 and 17 we report respectively check (i) and (ii). The tour is valid (line 20).

Listing 2.6: An example of distances between two deliveries

```
1  SERVICES DONE IN EACH STOP:
2    nr00 depot
3    nr01 delivery (EMI) (**)
4    nr02 delivery (**)
5    nr03 depot
6  PARAMETERS:
7    DISTANZA_CONSEGNE_EMI=500
8  VALIDATION:
9    dist(nr01, nr02) = 55 km
10   if 500 > 55 km  THE CHECK IS OK
11   else THE CHECK IS NOT_OK
```

Listing 2.7: An example of distances between delivery and pickup

```
1  SERVICES DONE IN EACH STOP:
2    00 depot
3    01 delivery
4    02 delivery (EMI) (**)
5    03 pickup (EMI)  (**)
6    04 pickup
7    05 depot
8  PARAMETERS:
9    DISTANZA_PRIMA_PRESA_EMI=80
10   PROVINCIE_SENZA_LIMITI_PRESA_DA_EMI_A_TOS
11   PROVINCIE_SENZA_LIMITI_PRESA_DA_EMI_A_LIG
12   FATTORE_RIENTRO_EMI=1.3
13 VALIDATION:
14   dist(nr02, nr03) = 49 km
15   dist(nr2, depot) = 162 km
16   dist(nr3, depot) = 131 km
17   (i) exists PROVINCIE_SENZA_LIMITI_PRESA_DA_EMI_A_EMI (FALSE)
18   (ii) 80 > 49 (FALSE)
19   (iii) 49 + 131 > 162 × 1.3 (FALSE)
20
21   if (i == TRUE) then  THE CHECK IS OK
22   else if (ii == TRUE) and (iii == TRUE) then THE CHECK IS NOT_OK
23   else THE CHECK IS OK
```

Listing 2.8: An example of distances between pickups

```
1  SERVICES DONE IN EACH STOP:
2    nr00 depot
3    nr01 delivery
```

```
4    nr02 delivery
5    nr03 pickup (EMI) (**)
6    nr04 pickup (**)
7    nr05 depot
8  PARAMETERS:
9    DISTANZA_PRESE_SUCCESSIVE_EMI=500
10   FATTORE_RIENTRO_PRESE_SUCCESSIVE_EMI=1.2
11 VALIDATION:
12   dist(nr03, nr04) = 31 km
13   dist(nr3, depot) = 131 km
14   dist(nr4, depot) = 80 km
15
16   (i) 31 > 500 (FALSE)
17   (ii) 31 + 80 > 131 × 1.2 (FALSE)
18
19   if (i == TRUE) and (ii == TRUE) then THE CHECK IS NOT_OK
20   else THE CHECK IS OK
```

2.3 Cost function

The cost computation of a tour receives as input the locations visited during the tour and returns, if the tour is feasible, the cost of the tour. The main data structures for the cost computation are $tratta$ and $tariffario\_prese$. In this section, we first describe these data structures and then we present the cost function pseudocode.

Listing 2.9: An example of tratta

```
1 from depot to  LAZ cost 713 euro
2  0-1 stop: 30 euro each stop
3  2-4 stop: 35 euro each stop
```

Listing 2.9 reports an example of a tratta. A vehicle has at least one tratta. The tratta contains both information to compute the cost of a tour (i) and to check its feasibility (ii). Line 1 means that the vehicle can deliver items to location sites in Lazio with a fixed cost of 713 euro. Line 2-3 encode the variable part of the cost of the tratta: the variable cost is of 30 euro for the first stop (line 2) and 35 for the second until the fourth stop (line 3). For instance, a tour doing 1 delivery in Lazio costs $713 + 1 \times 30 = 743$, a tour doing 2 deliveries in Lazio costs $713 + 1 \times 30 + 1 \times 35 = 778$, a tour doing 3 deliveries in Lazio costs $713 + 1 \times 30 + 2 \times 35 = 813$ and finally a tour doing 5 deliveries in Lazio results unfeasible according to the tratta (line 3 encodes that the maximum amount of stops doable should be ≤ 4).

Listing 2.10: An example of tariffario_prese

```
until 2 pickup 250 euro + 30 euro each additional pickup
```

Linsting 2.10 reports an example of tariffario_prese. Its meaning is: vehicles using it can collect items, it has a fixed cost of 250 euro ; if the tour using it has more then 2 stop there is an additional charge of 35 euro for each additional stop.

Listing 2.11 reports the pseudo-code of the cost function. From line 9 to 43 we describe the selection of the tratta to be used to compute the cost of the tour. The function HasTratta (line 16) checks if the item can be served using the available tratte (if the item cannot be served by any tratta the tour is unfeasible and ComputeCost returns -1). The function GetMinTrattaFeasible (line 19) return the tratta feasible with the minimum fixed cost. From line 43 until the end we report the use of the seleced tratta to cumpute the cost. The function ComputeCost (line 48) uses a tratta to compute the cost (for instance, the cost of a tour doing 3 stops using tratta reported in Listing 2.9 is $713 + 1 \times 30 + 2 \times 35 = 813$ *euro*). The function ComputeCostTariffarioPrese does the same using the tariffario_prese (for instance the cost of a tour doing 5 stops using tariffario_prese reported in Listing 2.10 is $250 + 3 \times 30 = 340$ *euro*).

Listing 2.11: Cost function pseudocode

```
1  ComputeCost(List<Tratta> tratte,
2              List<Tariffario_prese> tariffario_prese,
3              List<IItem> items) {
4
5     //maxtratta will contain the tratta selected
6     //to serve the items
7     Tariff maxtratta ← null;
8
9     if ((items are only deliveries) ||
10        (items are deliveries and pickup)) {
11
12       foreach (IItem item in items) {
13          //the check is done only for delivery part of tour
14          if (item is a pickup) break;
15
16          if (HasTratta(tratta, item)) {
17             //mintratta is the tratta with the lower
18             //cost for the item
19             Tratta mintratta ← GetMinTrattaFeasible(tratte, item);
20
21             //maxtratta is the mintratta with the
22             //higher cost of the tour
23             if ((maxtratta == null) ||
24                 (mintratta.Cost() > maxtratta.Cost()))
25                maxtratta ← mintratta;
26          }
27          else
28             //every delivery must have a tratta,
```

```
29              //return -1 means unfeasible tour
30              return -1;
31          }
32      }
33      //for pickup is not mandatory have a tratta
34      else if (items are only pickup)  {
35
36          if (HasTratta(tratte, item)){
37              Tariff mincost ←GetMinTrattaFeasible(tratte, item);
38
39              if ((maxtratta == null) ||
40                  (mincost.Cost() > maxtratta.GetCost()))
41                  maxtratta ← mincost;
42          }
43      }
44
45      if (maxtratta != null)
46          //we use the tariff with higher fixed cost
47          //to compute the cost of the tour
48          return ComputeCost(maxtratta);
49      else
50      {
51          //no tariff available (items are only pickup)
52          //then we try to use tariffario prese
53          res = ComputeCostTariffarioPrese(tariffario_prese, num_of_pu);
54          if (res == -1)
55              //tour not feasible in according with tariffario prese
56              return -1;
57          else
58              //return cost computed with tariffario prese
59            return res;
60      }
61  }
```

Chapter 3

Problem modelling

The optimal solution of our VRP consists on a daily plan serving all customers with minimal cost. Serve a customer means deliver, collect or pickup-and-deliver its goods. The goods of a customer can be aggregated becoming an item. A daily plan is made of tours. A tour has a cost, is achievable by a vehicle type and serves items. Each vehicle can do at most one tour.

This problem can be mapped to a Set Covering Problem (SCP) where tours are sets and elements to be covered are items. The mathematical formulation of our problem is the following:

$$minimize \sum_{p \in P} \sum_{k \in \Omega^p} c^k z^k + \sum_{q \in Q} d_q y_q \tag{1}$$

$$\sum_{p \in P} \sum_{k \in \Omega^p} x_q^k z^k + y_q >= 1 \qquad \forall q \in Q \tag{2}$$

$$\sum_{k \in \Omega^p} z^k <= n_p \qquad \forall p \in P \tag{3}$$

$$z^k \in \{0,1\} \quad \forall p \in P, \quad \forall k \in \Omega^p. \tag{4}$$

In this model, commonly referred as Master Problem (MP), P is the set of vehicle types and n_p represents the maximum number of vehicles of type p available per day. Q is the set of items to be delivered. Ω^p is the set of tours that can be assigned to a vehicle of type $p \in P$ in one day. Each coefficient x_q^k is 1 if item q is delivered along tour $k \in \Omega^p$, 0 otherwise. For each tour $k \in \Omega^p$, the variable z^k takes value 1 if tour is selected, 0 otherwise. For each item $q \in Q$, y_q takes values 1 if the item is not delivered, 0 otherwise. Covering constraints (2) impose that each item is served. Constraints (3) limit the number of tours assigned to vehicles of the same type.

The objective function (1) minimizes the total traveling costs plus the loss of revenue induced by a non delivered item d_q.

The master problem model may contain a number of variables, which grows exponentially with the size of the instance. To compute a valid lower bound, we recour to column generation. In particular, we relax integrality conditions on binary variables and consider a restricted master problem (RMP). Initially, the set of tour is empty but feasible solution is ensured by y variables. The cost d of the exclusions should be big enough to push the algorithm to find profitable tours which are dinamically generated solving pricing subproblems one for each vehilce type P.

The Pricing Problem At each column generation iteration the linear relaxation of the RMP is solved, and we search for new columns with negative reduced cost. The reduced cost of each column $k \in \Omega^p$ is:

$$\overline{c}^k = c^k - \sum_{q \in Q} \pi_q x_q^k - \gamma_p$$

where π_q is the nonegative dual variable associated to the qth constraint of the set (2) and γ_p is the nonpositive dual variable associated with the pth constraint of the set (3).

Hence, instead of explicitly computing the reduced cost of all the variables in the problem,we solve a pricing problem one for each vehicle type $p \in P$. If columns with negative reduced cost are found, they are inserted into the RMP and the process is iterated; otherwise, the optimal solution of the linear relaxation of the RMP is also an optimal solution of the linear relaxation of the MP.

Chapter 4

The algorithms

4.1 A column generation based heuristic with measure of quality

We devise an algorithm which combines three optimization techniques to produce heuristically VRP solutions with a measures of their quality.

Components Our algorithm is made of three components: an Ant Colony System (ANT), a Column Generation algorithm (CG) and a general purpose MIP solver (S). Figure 4.1 reports the components and their interactions. Each component is represented by a box. Ellipses represent input ant output.

ANT is an Ant Colony System implementation (Gambardella et al., 1999). It has been provided by AntOptima which is the software planning tools mediating our relations with the logistic company. ANT produces rapidly (around 10 minutes) heuristic solutions. Current implementation of ANT optimizes the total kilometric length of the tours instead of the total cost of the plan (euro). The role of ANT within our algorithm is to provide a starting solution (UB) to be used by CG.

CG is the column generation algorithm described in Section 3 which is first initialized with the solution provided by ANT and then optimally solved. At optimality, the objective value of the RMP's linear relaxation represents a tight lower bound (LB) of our problem and its columns are used to form a mixed integer linear problem (MIP).

Finally, the MIP is solved by a general purpose commercial solver S. The solution of the MIP is a plan (a set of tours) and its quality can be mesured using the LB computed by CG.

Components ANT and S are used as black-boxes, while CG has been designed and implemented by us. For this reason we provide a more in dept description

of this component.

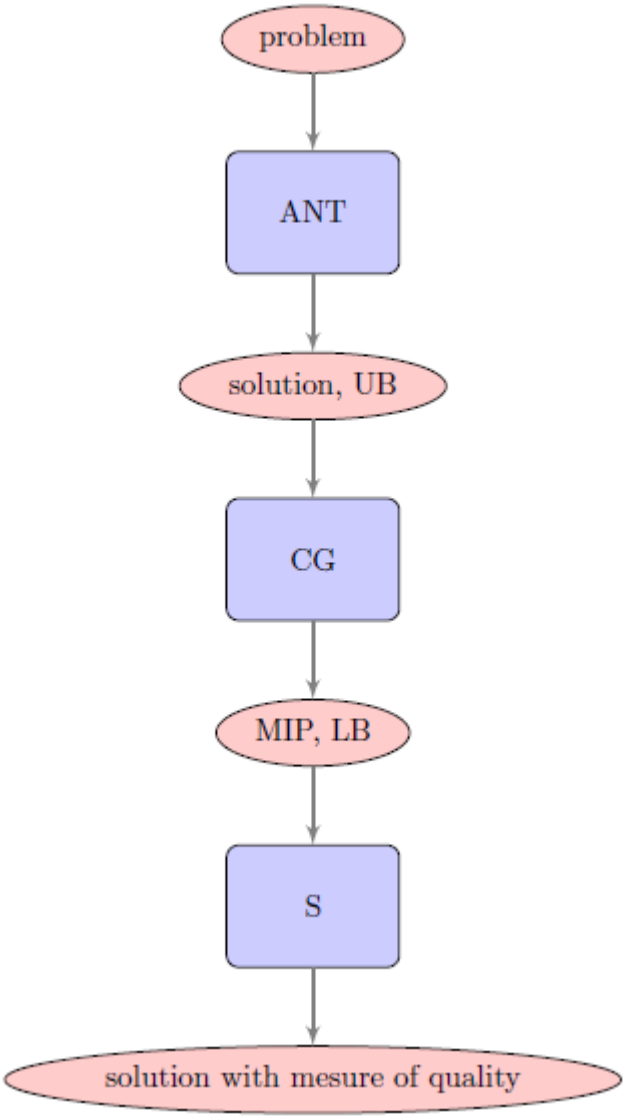


Figure 4.1: Algorithm components and interactions

Column Generation (CG) Figure 4.2 reports the high level flow chart of CG. Boxes are the sub-routines within CG, the ellipse on the top represents its input and the ellipse on the bottom is its output.

The input is a feasible solution previously computed by ANT and the corresponding UB. This solution is a set of tours which are used as initialization

columns of the RMP. The column generation algorithm works as follows: the linear relaxation of the RMP is solved, dual variables are collected and then multiple-pricing algorithms are solved to find columns with negative reduced cost. If columns with negative reduced cost are found they are inserted into the RMP and the process is iterated, otherwise the iteration finishes and the RMP is optimally solved. We obtain a lower bound (LB) of our problem (which is the objective function value of the linear relaxation of the RMP) and we can create the corresponding MIP. The MIP is an RMP with the integrality constraints on the tours selection, it is a model solvable by a generic commercial solver.

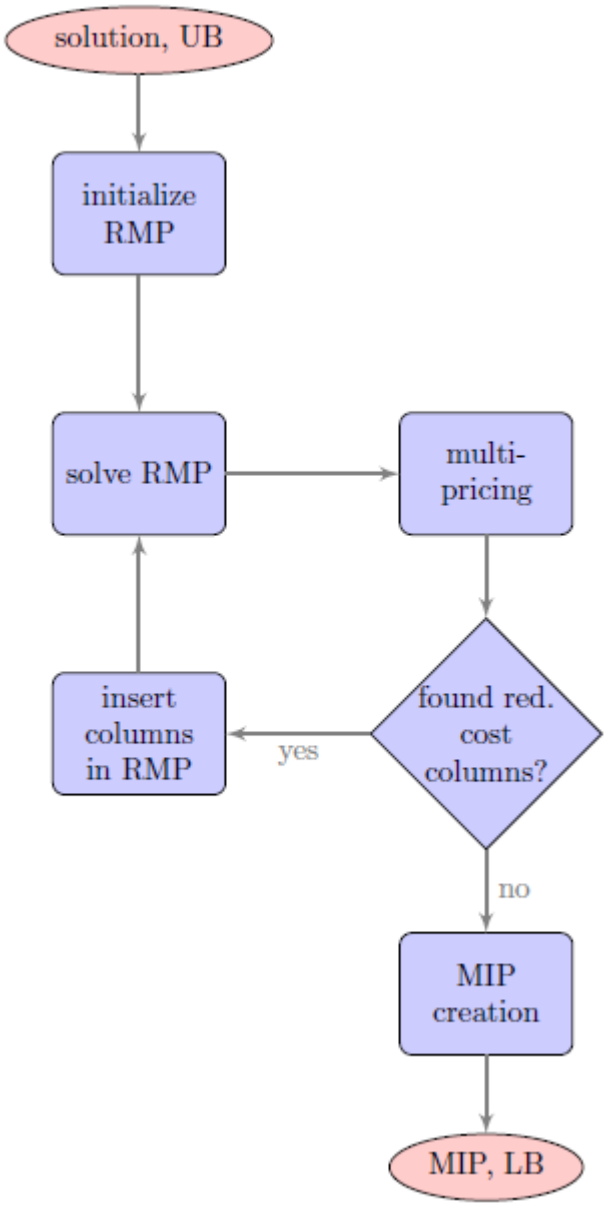


Figure 4.2: Routines and interactions within CG

Multiple-pricing The box "multi-pricing" of Figure 4.2 represents the pricing step. Its role is to find columns that price out profitability (i) or to prove that no one exists (ii). This problem is NP-hard (its formulation and algorithmic resolution will be reported on the second part of this chapter, see Section 4.2). Our implementation combines 3 pricing algorithms: a greedy pricing (G), an heuristic pricing (DPH) and an exact pricing (DPE). DPE solves the pricing problem to optimality, while G and DPH look heuristically for good solutions. The advantage of G and DPH compared to DPE is their speed. A multi-pricing approach allow to deal more efficiently with the complexity of the problem, in fact we use G and DPH to fastly find columns for each vehicle type P, while DPE is executed only to prove the optimality of the solution (ideally DPE is solved once for each vehicle type $p \in P$).

Listing 4.1 reports the pseude-code of the devised approach. We see the column generation cycle which is a while loop (lines 2-30) cycling until reduced cost columns are found. Pricing routines look for reduced cost columns (lines 7, 13 and 20). If a pricing routine succeds, variable $foundredcostcols$ takes value 1, the column are inserted into the RMP (line 26), the RMP is solved (line 27) and the process is iterated; otherwise the iteration finishes.

Listing 4.1: Multiple-pricing pseudo-code

```
1  bool foundredcostcols = 1;
2  while (foundredcostcols == 1)
3  {
4    foundredcostcols = 0;
5
6    foreach (p in P) {
7      foundredcostcols = G(p);
8      if (foundredcostcols == 1) exit from foreach loop;
9    }
10
11   if (foundredcostcols == 0) {
12     foreach (p in P) {
13       foundredcostcols = DPH(p);
14       if (foundredcostcols == 1) exit from foreach loop;
15     }
16   }
17
18   if (foundredcostcols == 0) {
19     foreach (p in P) {
20       foundredcostcols = DPE(p);
21       if (foundredcostcols == 1) exit from foreach loop;
22     }
23   }
24
25   if (foundredcostcols == 1) {
```

```
26      insert found reduced cost columns in RMP;
27      solve RMP;
28    }
29  }
```

4.2 The pricing alghoritm

The pricing problem can be modeled as a resource constrained elementary shortest path problem (RCESPP). The RCESPP is the problem of finding the minimum cost elementary path from a node s to a node t of a given graph such that the overall amounts of resources consumed do not exceed some given limits; resources are consumed when visiting nodes or traversing arcs. A graph $G(V,A)$ is given: its vertex set V is made by N vertices representing items and two special vertices s and t representing the depot. Let N indicate the set of the items; hence we have $V = N \cup \{s,t\}$. A non-negative integer cost c_{ij} is associated with each arc $(i,j) \in A$ and these costs satisfy the triangle inequality. A non-negative prize π is associated with each vertex $i \in N$ and a non-negative cost γ_p is associated with the depot. A vehicle must go from s to t, visiting a subset of the other vertices; no cycles are allowed. The objective is to minimize the cost, given by the sum of the costs of the arcs traversed minus the sum of the prizes collected at the vertices visited.

Because there are non-negative costs c associated with the arcs and non-negative prizes π associated with the vertices, negative cost cycles can occur. Therefore the requisite that the path must be elementary does not come for free from cost minimization but it must be explicitly enforced. When the underlying graph have negative cost cycles, the resource constrained elementary shortest path problem is strongly NP-hard (Dror, 1994).

The basic dynamic programming approach to the RCESPP is based on the algorithm devised by Desrosiers et al. (1981) for the RCSPP. It is an extension of the Bellman-Ford algorithm with the addition of resource constraints.

The algorithm assigns states to each vertex: each state of vertex i represents a path from s to i . Each state has an associated resource consumption vector R and each component of R represents the consumption of a different resource along the path. Each state has an associated cost C and the optimal solution is given by the minimum cost state associated with t. Different states associated with the same vertex i correspond to different feasible paths reaching i. Hence states are represented by a label of the form (R,C,i). The dynamic programming algorithm repeatedly extends states to generate other states. When a state (R,C,i) is extended to generate another feasible state (R',C',j), the cost and the resource consumption vector of the new state must be computed and those

states for which one or more components of R' exceed the available capacity are fathomed. The cost is initialized at 0 at vertex s and it is updated considering dual variables.

The effectiveness of the dynamic programming algorithm outlined above heavily relies upon the possibility of fathoming feasible states that cannot lead to an optimal solution. For this purpose suitable dominance tests are always performed when states are extended, so that the algorithm only records non-dominated states. Each state is represented by a label, that is a tuple (S, R, C, i), where S is a vector indicating the vertices already visited, R is a vector indicating the consumption of resources, C is the cost and i is the last reached vertex. The dominance test between two states, or labels, is the following. Let (S_1, R_1, C_1, i) and (S_2, R_2, C_2, i) be the labels of two states associated with vertex i . Then the former dominates the latter if:

$$S_1 \leq S_2 \tag{DS}$$
$$R_1 \leq R_2 \tag{DR}$$
$$C_1 \leq C_2 \tag{DC}$$

and at least one of the inequalities is strict. Extended states are not deleted, because they can be useful to dominate other states not yet generated. This implies to keep all non-dominated states in memory, but allows one to recognize dominations earlier than they would be if extended states were canceled.

Dynamic programming exact (DPE) DPE is our implementation of the algorithm described above. It is the exact pricing we use to prove the optimality of the RMP.

Listing 4.2 reports the pseudo-code of the implemented dynamic programming algorithm. We use following data structures: $items$ ecodes the vertices of the undelying graph, $Label$ encodes a state and $label\_to$ is an array of lists of labels (each vertex has an associated list of labels representing paths from s to itself). In lines 1-4 we instantiate the datastructure $label\_to$. In lines 6-8 we fill $label\_to$ adding the path (s, i) to each vertex $i \in items$. E.g. the initialized variable $label\_to$ of a graph made of three vertices $items = \{i_0, i_1, i_2\}$ is: $label\_to[0] = \{[s, i_0]\}$, $label\_to[1] = \{[s, i_1]\}$ and $label\_to[2] = \{[s, i_2]\}$. In lines 10 we report the loop iterating over open labels. Each detected open label is extended toward others items within the network (line 12). E.g. the extension of label $[s, i_1, i_2]$ toward item i_3 means to join i_2 with i_3 and the result is the label $[s, i_1, i_2, i_3]$. Once the label has been extended we check the feasibility of the resulting path according with the constraints of the problem (line 13). If $newlabel$ encodes a feasible path we test the dominance. The dominance is

tested according to inequalities DS, DR and DC. We test if *newlabel* is dominated by existing labels in $label[i]$ (lines 14-18). If an existing label dominates *mylabel*, *mylabel* is fathomed and we iterate on next item (line 11), othewise we do the opposite which is to check if *mylabel* dominates one of the existing (lines 19-23). If an existing label is dominated by *mylabel*, the existing label is fathomed (it is removed from $label\_to$ (label 21)). At this point we add *mylabel* to the list of labels of node i (line 25). When we tried to extend *openlabel* toward each item we close it (line 27) and we go to the next open label. The Algorithm ends when no open labels exist in $label\_to$.

Listing 4.2: Dynamic programming algorithm pseudo-code

```
1  List<Label>[] labels_to = new [items.count];
2  for (int i = 0; i< items.count; i++) {
3    labels_to[i] = new List<Label>();
4  }
5
6  for (int i = 0; i< items.count; i++) {
7    labels_to[i]= create a label of path (s,i);
8  }
9
10 while (openlabel  in labels_to) {
11   for (int i = 0; i< items.count; i++) {
12     Label mylabel = openlabel +  items[i];
13     if mylabel is feasible {
14       for (int i = 0; i< items.count; i++) {
15         if (labels_to[i] dominates mylabel) {
16           go to line 13;
17         }
18       }
19       for (int i = 0; i< items.count; i++) {
20         if (mylabel dominates labels_to[i]) {
21            labels_to.remove(i);
22         }
23       }
24     }
25     labels_to[i].add(mylabel);
26   }
27   openlabel.close();
28 }
```

Dynamic programming heuristic (DPH) The dynamic programming algorithm reported in Listing 4.2 can be used heuristically. The heuristic consists on a more aggressive fathoming strategy to reduce the search tree. This is done replacing

the dominance tests reported in (DS, DR and DC) with:

$$r_1 \leq r_2 \tag{Dr}$$
$$C_1 \leq C_2 \tag{DC}$$

where r is a subset of the resources. In our implementation we considered only two resources: the number of item served by the label and the time consumption associated with it.

Greedy pricing (G) Listing 4.3 reports the pseudo-code of the greedy algorithm we used within our column generation procedure. Datastructure *items* econdes the vertices of the undelying graph. The algorithm first sorts the items considering the corresponding dual values π_q in a decreasing order (line 1), then it creates a tour (line 3) and iteratively tries to add items to the tour (line 6). If the resulting tour is not feasible the item is removed (line 8), otherwise it is kept into the tour and the algorithm considers the next item.

Listing 4.3: Greedy algorithm pseudo-code

```
1  List<IItem> sorteditems  = items.SortByDualDesc;
2
3  Tour mytour = new Tour();
4
5  for (int i = 0; i< sorteditems.count; i++) {
6    mytour = mytour + sorteditems[i];
7    if (mytour is not feasible) {
8      mytour = mytour - sorteditems[i];
9    }
10 }
```

4.3 An incremental heuristic (IH)

We devised an incremental heuristic which extends the standard column generation procedure reported in Figure 4.2.

The basic idea is the following: we start generating columns for a subnetwork using pricing algorithms (DPH and G). When heurisitc pricings stop finding columns on the subnetwork, we increase its dimension and we start again the column generation algorithm. We repeat this procedure until the subnetwork is equal to the original one. At this point we execute also the exact pricing (DPE) to prove the optimalty of the solution.

Figure 4.3 shows the method. H represents the size of the subset, its value must be $0 > H \leq 1$ (e.g. $H = 0.5$ means that the subset is 50% of the original;

$H = 1$ means a subset equals the original). The initialization of H is done at the beginning of the procedure (box "initialize RMP and H"), and it consists in assigning an initial value to H. Parameter ΔH is the increment step performed when both "found red. cost column?" and "H==1" returns "no", this is when no columns are found and the problem is still reduced. Boxes "pricingE(H)" and "pricingH(H)" represent respectively the pricing step using the exact algorithm and the pricing step using heuristic algorithms only.

Network reduction The subset of the network is obtained through an heuristic procedure. The procedure reduces the dimension of the problem removing arcs from the underlying network. The result is a subset of the problem. Listings 4.4 reports the pseudo-code of the procedure. Let $G' = (V', A')$ be the reduced graph representing the subset of the problem and H $(0 > H \leq 1)$ is the desired size of the subset. Let $\delta^-(S \in V)$ be the set of arcs with the tail in the set S. The idea is to remove arcs from A' until their number $|A'|$ is less or equal the desired number of arcs which is $|A| \times H$ (line 2).

Let v_{max} be the vertex with the highest number of outgoing arcs within V' and $a_{v_{max}}$ be the longest arc within outgoing arcs of v_{max}: $a_{v_{max}}$ is removed from A' and the process iterates.

Listing 4.4: Network reduction pseudo-code

```
1 G' = G;
2 while (|A'| > H × |A|) {
3    v_max = max_{v∈V'} |δ⁻({v})|
4    a_{v_max} = max_{a∈δ⁻({v_max})} lenght(a)
5    A' = A' \ {a_{v_max}}
6 }
```

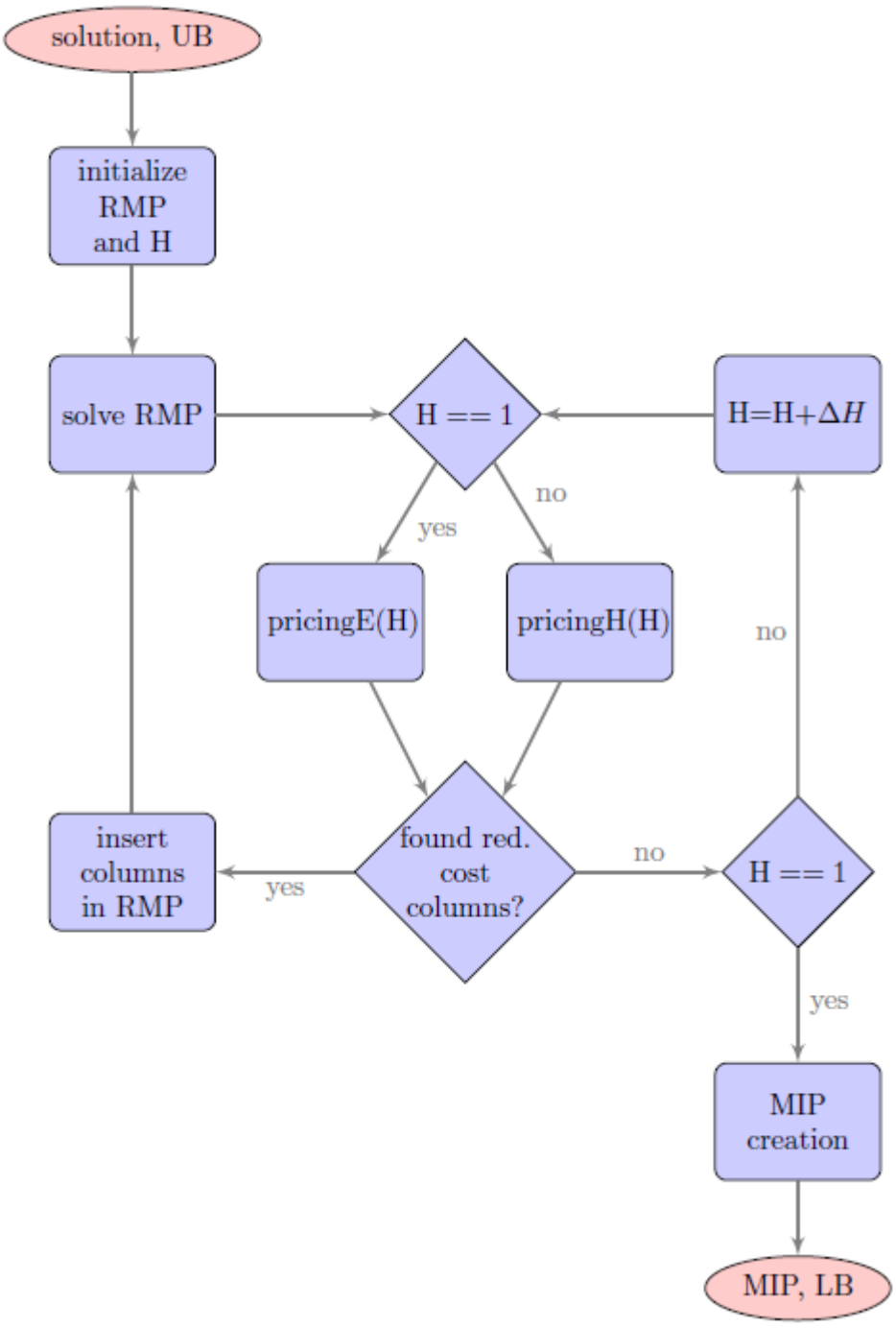


Figure 4.3: Routines and interactions within CG implementing the incremental heuristic

Chapter 5

Experimental Results

We propose a test campaign which shows the effectiveness of the algorithm proposed in Chapter 4.

Section 5.1 reports the experimental results of the standard column generation algorithm. We solve 14 instances and we provide solutions with measure of their quality. In Section 5.2, we propose a possible application of our algorithm to a real-world situation. We assign time-limits to each of its component in order to provide solution whithin a given amount of time. This approach allows to handle instances of considerable dimension within a limited amount of time. Consequently, the method becomes heuristic (the measure of the solution's quality is no more guaranteed) but its application becomes more attractive from the logistic company's point of view. The maxium amount of time to provide a solution according with the requirements of our logistic partner is 30 minutes.

Pre-processing - candidate list To speed-up the algorithm, we perform some pre-processing. We define candidate list as the list of vertices reachable by a given vertex within the transportation network. Each vertex has an associated canidate list. The pricing algorithm is restricted to evaluate the label extensions from a vertex to the subset of reachable vertices. The idea is to use some of the constraints reported in Section 2.2 to pre-process our network to reduce the size of the candidate lists. The constraints used to pre-process the network are time windows, primary secondary zones and maximal distance between stops. This pre-processing allows to reduce the average size of the candidate list by 90%, the result is a considerable speed-up of the algorithm (more than 2 times faster).

Pre-processing - vehicles aggregation Vehicle types are described by several characteristics (the availability of an hydraulic lift, the availability of a transpallet,...) and they encode information used to compute the cost of the tours. Our

customer defines those features as a single vehicle setting which is encoded in the instance of the problem. An instance of our problem contains in average more then 140 vehicles. Since the column generation algorithm reported in Chapter 3 requires the resolution of a pricing problem for each vehicle types $p \in P$, a big $|P|$ affects negatively the efficiency of the method. From an analysis of the datastructures describing the vehicle types, we realized that many vehicles have exactly the same settings. We can then aggregate vehicles into groups of vehicles with the same characteristics. The aggregation, in average, allows to reduce $|P|$ by 60%.

Instances For our experiments we used real world data provided by the commercial customer. We used 14 instances which are 14 plans, each plan is a set of items (deliveries, pickup and pu-and-del) for a specific day. Datailed informations concerning those datasets are reported in Table 5.1. The first column of the table reports the name of the instance which consists on the date of the plan, the second column reports the corrisponding day of week, then we report the number of vehicle types, the next column represents the arcs within the transportation network defined by the problem. The last 4 columns reports the number of items and their reparition between delivery, pickup and pu-and-del. The average number of items within an instance is 160, the biggest instance is the 08.03.2010 and contains 201 items, the smallest instance is the 07.10.2010 and contains 115 items.

Typical solution The average number of tours in a solution is 37 (from 29 up to 46). A tour serves in average 4.4 items (from 1 up to 12). A tour costs in average 830 euro (from 150 up to 2680).

Technologies All tests were performed on a PC equipped with an Intel Core i7 2.67 GHz 2 Cores processor with 3 GB RAM. The column generation algorithm were coded in c# 2.0 and the solver used was Gurobi 4.0.

Instance	day of week	vehicles	arcs	items	deliv.	pickup	pu-and-del
08.03.2010	Wednesday	77	4509	201	147	50	4
10.03.2010	Wednesday	75	4611	189	134	52	3
11.03.2010	Thursday	77	4805	191	150	39	2
04.05.2010	Tuesday	74	5513	186	124	57	5
05.05.2010	Wednesday	75	5286	200	148	50	4
14.06.2010	Monday	77	4062	182	132	47	3
06.08.2010	Friday	86	2419	141	96	43	2
20.09.2010	Monday	86	3557	173	129	43	1
21.09.2010	Tuesday	85	2477	138	96	41	1
22.09.2010	Wednesday	87	2388	143	106	36	1
23.09.2010	Thursday	85	2707	142	95	45	2
24.09.2010	Friday	87	2729	148	107	40	1
07.10.2010	Friday	86	1572	115	88	26	1
13.10.2010	Wednesday	89	1480	120	95	24	1

Table 5.1: Real-world instances

5.1 Results

We used the algorithm reported in Chapter 4 to compute solutions with the mesure of their quality.

We noticed that the dimensions of real-world instances of Table 5.1 are considerable and their resolution can not be done within a reasonable amount of time. Hence we reduced instances of Table 5.1. The reduction has been done removing the 70% of the arcs from the trasportation network of each real-world instance. The strategy used to perform the reduction is reported in Chapter 4 at Listings 4.4 (we set parameter H equals 0.3). This preserves the size of the original instance in terms of items and vehicle types. Consequently, the solution is comparable with that of the full instance.

Table 5.2 reports the produced results. The first column reports the name of the instance (they correspond to entries of Table 5.1 which were here reduced). The second column contains the number of arcs in the instance. The next 4 columns report information about the execution time of the algorithm: the total time (in seconds) and its repartinition between the 3 components. The next columns reports the lower bound computed by CG and the cost (in euro) of the solution. The last column reports the gap % which is the measure of the quality of the solution, it is computed using the followings formula $(cost \in -LB)/LB \times 100$.

From the examination of Table 5.2 we observe that the algorithm provides solutions in a reasonable amount of time for all instances but one ($11.03.2010_r$). The size of the instance affects the total time needed to provide a solution and its repartition between the components: CG and S need more time when the instance is big, while the execution time of ANT is constant because it is managed with a time-limit of 600 seconds. The MIP solver S is able to produce the optimal solution in less than 80 seconds for 12 instances out of 14 and needs around 1 hour for the other 2. As expected, for those instances, we remark that CG procedure is the most time consuming. The best result has been obtained with instance $13.10.2010_r$ (0.52%) while the worst is that of the $23.09.2010_r$ (4.28%). The average gap is of 2.47% , this confirms that the algorithm produces good quality solutions.

Instance	arcs	t_{tot}	t_{ANT}	t_{CG}	t_S	LB	cost €	gap %
$08.03.2010_r$	1352	905	600	295	10	42032.68	42561.77	1.26
$10.03.2010_r$	1383	1988	600	1355	33	36518.78	37444.37	2.53
$11.03.2010_r$	1441	11695	600	11015	80	25112.46	25724.77	2.44
$04.05.2010_r$	1653	5913	600	2400	2913	30012.60	30997.12	3.28
$05.05.2010_r$	1585	5965	600	1500	3865	39304.94	40464.44	2.95
$14.06.2010_r$	1218	927	600	317	10	40385.00	40994.53	1.51
$06.08.2010_r$	725	810	600	200	10	21050.27	21908.53	4.08
$20.09.2010_r$	1067	768	600	155	13	37987.00	39016.46	2.71
$21.09.2010_r$	743	729	600	124	5	28479.42	29129.80	2.28
$22.09.2010_r$	716	746	600	138	8	27437.03	28327.71	3.25
$23.09.2010_r$	812	888	600	275	13	24633.62	25688.69	4.28
$24.09.2010_r$	818	847	600	235	12	22182.57	22649.53	2.11
$07.10.2010_r$	471	633	600	32	1	18190.28	18448.32	1.42
$13.10.2010_r$	444	627	600	26	1	22690.75	22809.48	0.52

Table 5.2: Algorithm solutions with measure of quality

Setting	ANT_{tl}	CG_{tl}	S_{tl}
s_30_0_0	30	0	0
s_10_5_15	10	5	15
s_10_10_10	10	10	10
s_10_15_6	10	15	5
s_0_20_10	0	20	10

Table 5.3: Time-limit settings

5.2 Real world instances resolution within 30 minutes

Our algorithm allows to to obtain solutions with a mesure of their quality. The payload of the method is the time required to provide that measure. Unfortunaltely real-world logistic companies operate with tight deadlines, the instances are big and the algorithm execution time becomes then a key-factor for its application. Dimension of real world instances and time constraint imposed by our logistic partner led us to limit the execution time of our algorithm. The result is then an heuristic method and the measure of the quality of the solution is not guaranteed anymore.

As reported in Chapter 4, we propose an optimization system made of 3 components: an Ant Colony System (ANT), a Column Generation algorithm (CG) and a general purpose MIP solver (S). Let ANT_{tl}, CG_{tl} and S_{tl} be the time-limits in minutes assigned respectively to components ANT, CG and S. Since the maximum execution time of the whole system accordingly with our logistic partner is 30 minutes, we have the following constraint: $ANT_{tl}+CG_{tl}+S_{tl} \leq 30$. We tested 5 time-limit settings, the detail of each setting is reported in Table 5.3. Morover, using the same time-limits, we compared results of the standard column generation procedure against the version implementing the incremental heuristic (IH) proposed in Chapter 4.

Our goal is to discover the setting which provides the best solutions and shows a stable behaviour (always produces a feasible solution).

Tables 5.4-5.8 report the results produced for each setting. Table 5.4 has only 3 column because the results were obtained through the setting s_30_0_0 which executes only component ANT for 30 minutes. The first column reports the name of the instance, the second reports the cost (in euro) of the solution. The last column is "Δ%" which reports the rate of difference between the cost of the solution against the best solution computed considering all the settings (e.g. a value of *3* means that the current solution costs 3% more than the best solution computed using the others settings). Tables 5.5-5.8 are organized as follows: the first column contains the name of the instance, the second column reports the

cost (in euro) of the solution computed by ANT at the corresponing time-limit. Columns 3-6 refer to standard column generation procedure (STD CG), while columns 7-10 refer to procedure implementing the devised incremental heuristic (CG WITH IH). Columns "cols" and "RMP" report the number of columns and the value of the RMP within component CG at time-limit. Columns "cost €" and "Δ%" report the final solution provided by our algorithm and its rate against the best solution. In order to highlight the best results we mark entries in **bold underlined** when they are the best and unique and in **bold** when more settings get the same best cost. Morover, in columns 5 and 9, we mark with an asterisk (*) the solutions of the MIPs which are proven optimal. Entries with (-) indicates that the algorithm was not able to produce a solution serving all the items within the time-limit.

Table 5.4 highlights 1 best solution which is also unique. There is a feasible solution for each instance but, in general, their quality is not good (solutions have Δ% up to 10%).

Table 5.5 highlights 1 best solution which is also unique and was obtained using STD CG. The value of the RMP obtained using STD CG is always less (from 1.1% to 7.6%) than that obtained using CG WITH IH. STD CG, at time-limit, generates more columns than CG WITH IH. In fact, the generated number of columns using STD CG is within a range going from 21295 to 116737, while their range using CG WITH IH goes from 7157 to 20063. As consequence, the solver S can solve easier MIPs generated using CG WITH IH than those generated using STD CG. In fact, S proved the optimality of 14 MIPs: only 2 of them were obtained by STD CG, while the others (12) were obtained by CG WITH IH. There is a feasible solution for each instance, their overall quality is not so good using STD CG (Δ% up to 8.3%), while is quite good (Δ% up to 4.4%) using CG WITH IH.

Table 5.6 highlights 6 best solutions: 2 of them were obtained by STD CG, while the others (4) were obtained by CG WITH IH. The value of the RMP obtained using STD CG is always less (from 0.4% to 2.9%) than that obtained using CG WITH IH. STD CG, at time-limit, generates more columns than CG WITH IH. In fact, the generated number of columns using STD CG is within a range going from 21837 to 135512, while their range using CG WITH IH goes from 8058 to 26369. As consequence, the solver S can solve easier MIPs generated using CG WITH IH than those generated using STD CG. In fact, S proved the optimality of 5 MIPs which were all produced by CG WITH IH. There is a feasible solution for each instance, their overall quality is not so good using STD CG (Δ% up to 8.3%) while is good, and also the best, using CG WITH IH (Δ% up to 2.2%).

Table 5.7 highlights 6 best solutions: 1 of them was obtained by STD CG, while the others (5) were obtained by CG WITH IH. The RMP computed using

STD CG is less (from 0.2% to 2.4%) then that of CG WITH IG for all instances but two (20.09.2010 and 07.10.2010 where the RMP computed using CG WITH IG is 0.1% less then that of STD CG). STD CG, at time-limit, generates fewer columns than CG WITH IH. In fact, the generated number of columns using STD CG is within a range going from 22315 to 136760, while their range using CG WITH IH goes from 9117 to 30025. As consequence, the solver S can solve easier MIPs generated using CG WITH IH than those generated using STD CG. In fact, S proved the optimality of 3 MIPs which were all produced by CG WITH IH. There is a feasible solution for each instance, their overall quality is not so good using STD CG (Δ% up to 8.3%) while is quite good using CG WITH IH (13 solutions have Δ% less then 1.4% and 1 solution has Δ% equals 5.6%).

Table 5.8 highlights 3 best solutions (which are also unique), they were obtained using CG WITH IH. The RMP computed using STD CG is less (from 0.2% to 1.6%) then that of CG WITH IG for all instances but one (13.10.2010 where the RMP computed using CG WITH IG is 0.1% less then that of STD CG). CG WITH IH, at time-limit, generates fewer columns than STD CG (the generated number of columns using STD CG is within a range going from 20321 to 168102, while their range using CG WITH IH goes from 12839 to 67646). As consequence, the solver S can solve easier MIPs generated using CG WITH IH than those generated using STD CG. In fact, S proved the optimality of 1 MIPs which were produced by CG WITH IH. This setting not always produces a feasible solution, this happens 6 times when the executed algorithm was the STD CG and 5 times when the executed algorithm was CG WITH IH. The overall quality of the solutions, when they exists, is instable. In fact, using STD CG we have 8 solutions with values of Δ% distributed in a range going from 0.1 % to 7.9% , while using CG WITH IH we have 9 solutions with values of Δ% going from 0.0 % to 7.1% .

Figures 5.1-5.2 are charts depicting two curves each one: the dashed line has been produced by STD CG, while the other has been produced by CG WITH IH. The data has been collected during the 15 minutes resolution of instance 08.03.2010 (we noticed that others instances have the same behaviour). Figures 5.1 reports the evolution of the RMP's linear relaxation objective function value, while Figure 5.2 reports the evolution of the number of columns within the RMP.

From the examination of Figure 5.1 we can observe the typical behaviour of the RMP and indetify the so called "tailing off" effect of column generation. Morover, comparing Figure 5.1 with Figure 5.2, we observe that after 15 minutes the gap between the RMPs is of +0.73% while the gap in terms of number of columns is of -62%. This confirms that IH is able to detect good columns in the early stages of the algorithm.

Table 5.9 aggregates the results of the tests. It contains a row for each table in 5.4-5.8. The first column identify the setting. Columns 2-6 refer to standard column generation procedure (STD CG), while columns 7-11 refer to procedure implementing the devised incremental heuristic (CG WITH IH). The column "cost €" reports the average cost of the solution, the column "cols" reports the average number of columns of the RMP at time-limit and column "RMP" reports its value. Column "mip*" reports the number of instances where the MIP has been optimally solved by S. Column "Best rate (%)" reports the percentage of best solutions considering only feasible solutions. The formula used to compute the "Best rate (%)" is the following: $best/solved \times 100$ where $best$ counts best results within the setting (costs marked in bold) and $solved$ counts provided feasible solutions (costs others then "-"). This value is used to indicate the overall quality of the solutions when they are provided.

From the examination of the results of the tests it is possible to draw some conclusions about the effectiveness and the performances of our algorithm:

Our algorithm allows to deal with real-world instances providing intereseting results within 30 minutes. Our algorithm is designed to provide a measure of the quality of the solution. Unfortunately with the restrictive setting imposed by the logistic company, the algorithm fails to produce such measure in most of the cases. Despite of that, we can estimate the quality of the solution:

1. we can closely approximate the quality of the solution considering the behavior of CG component of our algorithm depicted in Figure 5.1. We observe that after 15 minutes, CG is in a state of "tailing-off". In this state, the RMP value is close to the LB. Using RMP value to compute the gap of the solution of our algorithm (using the best setting) we obtain 3.27%.

2. we can evaluate the solution through a comparison with the solution obtained by running ANT for 30 minutes (in fact ANT can also be used as a standalone optimization system) which is 31185.19 euro (see Table 5.9 setting s_30_0_0). Solution computed by our algorithm, using its best setting, costs 29930.16 euro (see Table 5.9). The gain is of 1255.03 euro (-4%).

Incremental heuristic IH helps. Table 5.9 highlights the best solution obtained from our tests. The version of the algorithm used to obtain that solution combines setting s_10_10_10 and the incremental heurisitc IH we devised in Chapter 4. The produced solution costs 29930.16 euro. We propose a comparison between this solution and the best solution computed using the standard column generation procedure which is s_10_5_15 and costs 30348.30 euro:

- IH improves the final solution by -1.37% (29930.16 instead of 30348.30)

- IH RMP's value at time-limit is similar +0.73% (28971.72 instead of 28761.38) (see Figure 5.1)
- IH RMP's columns number at time-limit is smaller -62% (16898 instead of 44540) (see Figure 5.2).

These indicators suggests how a RMP's relaxation with few columns produces a MIP which can be solved more easily by the general purpose solver S. We can claim that, since we have time constraint and we have to balance the load of work between CG and S, an effective procedure (IH) which allow to intelligently reduce the number of columns generated by CG helps S and increases the performance of the algorithm.

Initialization helps the algorithm being more stable. The column generation algorithm can be initialized using known solution (just adding the known tours as columns into the RMP). Our experiments highlight how the initialization helps the algorithm being more stable. In fact, at the end of the optimization chain, we always have a feasible solution. We also observe that, when CG is not initialized, it is able to produce in few cases the best solutions thus the "Best %" value is the higher (see Table 5.9, setting s_0_20_10, CG WITH IH).

A comparaison between results of current section and results of Section 5.1 In Section 5.1 we executed our algorithm without time-limit to solve reduced instances (NTL_I$_r$). The used reduction strategy preserves the size of the original instance in terms of items and vehicle types. Consequently, solutions of NTL_I$_r$ are comparable with solutions computed in the current section (where we solve original instances using the 30 minutes heuristic algorithm).

The best average cost obtained in the current section is 29930.16 euro (see Table 5.9), while the average cost of solutions provided by NTL_I$_r$ is 30440.40 euro (see Section 5.1 Table 5.2).

The algorithm proposed in the current section solves instances of original size, within a fixed amount of time and provides better solutions (the average gain is 510.24 euro), while algorithm NTL_I$_r$ solves reduced instances, doesn't ensure a feasible solution within a reasonable amount of time (the average time needed is 34 minutes, but the resolution of instance 11.03.2010$_r$ takes more then 3 hours) and the provided solutions are worse. Despite, the big advantage of algorithm NTL_I$_r$ is the capability to provide the measure of the quality of the solution.

	ANT	
Instance	cost €	Δ%
08.03.2010	44043.84	*5.9*
10.03.2010	38734.42	*2.7*
11.03.2010	26194.80	*1.8*
04.05.2010	32177.13	*4.4*
05.05.2010	**40408.44**	*0.0*
14.06.2010	41673.51	*3.5*
06.08.2010	22388.53	*8.3*
20.09.2010	39198.41	*3.3*
21.09.2010	30027.80	*6.7*
22.09.2010	28862.74	*6.3*
23.09.2010	26578.65	*8.6*
24.09.2010	23781.58	*7.8*
07.10.2010	19093.28	*10.0*
13.10.2010	23429.48	*7.2*

Table 5.4: s_30_0_0 results

	ANT	STD CG				CG WITH IH			
		CG		S		CG		S	
Instance	cost €	cols	RMP	cost €	Δ%	cols	RMP	cost €	Δ%
08.03.2010	44138.80	28439	40801.99	**41580.95**	*0.0*	11426	41599.57	42230.76*	*1.6*
10.03.2010	38734.43	40572	35971.76	38734.43	*2.7*	14108	36688.84	38144.39*	*1.1*
11.03.2010	26124.80	116737	24676.35	26124.80	*1.6*	19372	25485.22	25754.78*	*0.1*
04.05.2010	32547.12	65116	29771.89	32547.12	*5.6*	18089	30299.39	31415.00*	*1.9*
05.05.2010	40637.43	63561	38480.70	40637.43	*0.6*	20063	39445.79	40564.43	*0.4*
14.06.2010	41643.53	41954	39012.08	41003.54	*1.8*	15370	39667.31	40844.49*	*1.4*
06.08.2010	22388.53	35932	19913.74	22388.53	*8.3*	11041	20458.19	21558.46	*4.3*
20.09.2010	39108.42	46246	36945.34	37998.47	*0.1*	14955	37373.16	38668.44*	*1.9*
21.09.2010	30076.83	28648	27118.45	28609.84	*1.7*	8755	27877.62	28929.79*	*2.8*
22.09.2010	29002.76	30591	26302.07	27732.80	*2.1*	9268	26964.80	27717.70*	*2.1*
23.09.2010	26778.66	31675	23833.85	24973.74	*2.0*	8337	24365.81	25168.69*	*2.8*
24.09.2010	23901.59	50340	21365.84	23051.75	*4.5*	12799	21836.70	22321.51*	*1.2*
07.10.2010	18928.30	25312	16969.07	17553.35*	*1.2*	10055	17545.76	18113.25*	*4.4*
13.10.2010	23619.46	21295	21496.18	21939.52*	*0.3*	7157	21747.61	22360.48*	*2.3*

Table 5.5: s_10_5_15 results

	ANT	STD CG				CG WITH IH			
		CG		S		CG		S	
Instance	cost €	cols	RMP	cost €	Δ%	cols	RMP	cost €	Δ%
08.03.2010	44138.80	30002	40700.62	42330.96	*1.8*	16740	40898.19	41840.92	*0.6*
10.03.2010	38734.43	43603	35833.58	38734.43	*2.7*	18586	36200.99	**37724.37**	*0.0*
11.03.2010	26124.80	135512	24504.89	26124.80	*1.6*	22475	25241.76	**25724.77***	*0.0*
04.05.2010	32547.12	74115	29352.37	32534.15	*5.6*	26125	29681.77	**30817.00**	*0.0*
05.05.2010	40637.43	67941	38251.23	40637.43	*0.6*	26369	38862.17	40637.43	*0.6*
14.06.2010	41643.53	43276	38819.58	**40263.51**	*0.0*	19965	39399.56	40634.48*	*0.9*
06.08.2010	22388.53	36568	19876.30	22388.53	*8.3*	13867	20053.38	**20673.48**	*0.0*
20.09.2010	39108.42	46741	36877.10	38078.54	*0.3*	17827	37100.84	38258.47*	*0.8*
21.09.2010	30076.83	28907	27088.88	30026.83	*6.7*	13265	27256.81	28349.77	*0.8*
22.09.2010	29002.76	31157	26206.71	27862.82	*2.6*	11631	26667.96	27427.74	*1.0*
23.09.2010	26778.66	32072	23782.09	24763.71	*1.2*	11651	24028.94	24818.63*	*1.4*
24.09.2010	23901.59	52107	21223.53	22479.67	*1.9*	15900	21527.99	22231.51	*0.8*
07.10.2010	18928.30	26251	16888.54	**17353.35**	*0.0*	14116	17179.61	17743.30*	*2.2*
13.10.2010	23619.46	21837	21412.22	21939.52	*0.3*	8058	21504.16	22140.43	*1.3*

Table 5.6: s_10_10_10 results

	ANT	STD CG				CG WITH IH			
		CG		S		CG		S	
Instance	cost €	cols	RMP	cost €	Δ%	cols	RMP	cost €	Δ%
08.03.2010	44138.80	40549	40549.32	44138.80	*6.2*	17403	40815.50	42000.89	*1.0*
10.03.2010	38734.43	44675	35814.53	38734.43	*2.7*	20639	36106.72	37773.34	*0.1*
11.03.2010	26124.80	136760	24384.10	25724.78	*0.0*	26716	24994.01	**25724.77***	*0.0*
04.05.2010	32547.12	74938	29316.98	32534.15	*5.6*	30025	29616.28	32547.12	*5.6*
05.05.2010	40637.43	68683	38224.67	40637.43	*0.6*	29996	38606.90	40637.43	*0.6*
14.06.2010	41643.53	43379	38802.92	40453.51	*0.5*	20538	39299.97	40364.46*	*0.3*
06.08.2010	22388.53	36858	19848.91	22388.53	*8.3*	14834	19968.88	20768.45	*0.5*
20.09.2010	39108.42	47282	36872.88	39108.42	*3.0*	19954	36827.31	**37958.51**	*0.0*
21.09.2010	30076.83	28907	27088.88	30026.83	*6.7*	15708	27144.36	28199.78	*0.2*
22.09.2010	29002.76	31226	26199.40	28052.84	*3.3*	13021	26472.84	**27157.73**	*0.0*
23.09.2010	26778.66	32072	23782.09	24763.71	*1.2*	12542	23952.59	**24478.65***	*0.0*
24.09.2010	23901.59	52380	21173.04	23191.61	*5.1*	16865	21364.27	**22191.55**	*0.6*
07.10.2010	18928.30	26290	16886.89	**17353.35**	*0.0*	15324	16979.25	17563.36	*1.2*
13.10.2010	23619.46	22315	21381.27	21939.51	*0.3*	9117	21355.85	22000.41	*0.6*

Table 5.7: s_10_15_5 results

	ANT	STD CG				CG WITH IH			
		CG		S		CG		S	
Instance	cost €	cols	RMP	cost €	Δ%	cols	RMP	cost €	Δ%
08.03.2010	-	39341	42125.57	43749.97	*5.2*	26283	42801.72	44549.94	*7.1*
10.03.2010	-	55064	-	-		27941	-	-	
11.03.2010	-	168102	-	-		67646	-	-	
04.05.2010	-	59568	30513.94	32116.17	*4.2*	39183	30855.69	32786.06	*6.4*
05.05.2010	-	83106	-	-		41322	-	-	
14.06.2010	-	45129	39286.19	41471.64	*3.0*	30646	39804.27	41167.45	*2.2*
06.08.2010	-	36989	19847.16	22308.61	*7.9*	23299	20036.33	20688.46	*0.1*
20.09.2010	-	62424	37190.52	39757.52	*4.7*	40838	37302.67	38457.45	*1.3*
21.09.2010	-	39379	27074.72	-		24172	27116.68	**28129.81**	*0.0*
22.09.2010	-	34120	-	-		16722	-	-	
23.09.2010	-	29415	-	-		21486	-	-	
24.09.2010	-	49190	21132.43	22061.67	*0.1*	26704	21472.74	**22060.57**	*0.0*
07.10.2010	-	23421	16911.33	17433.28	*0.5*	16899	17065.50	17663.31*	*1.8*
13.10.2010	-	20321	21389.55	21900.53	*0.2*	12839	21369.86	**21864.53**	*0.0*

Table 5.8: s_0_20_10 results

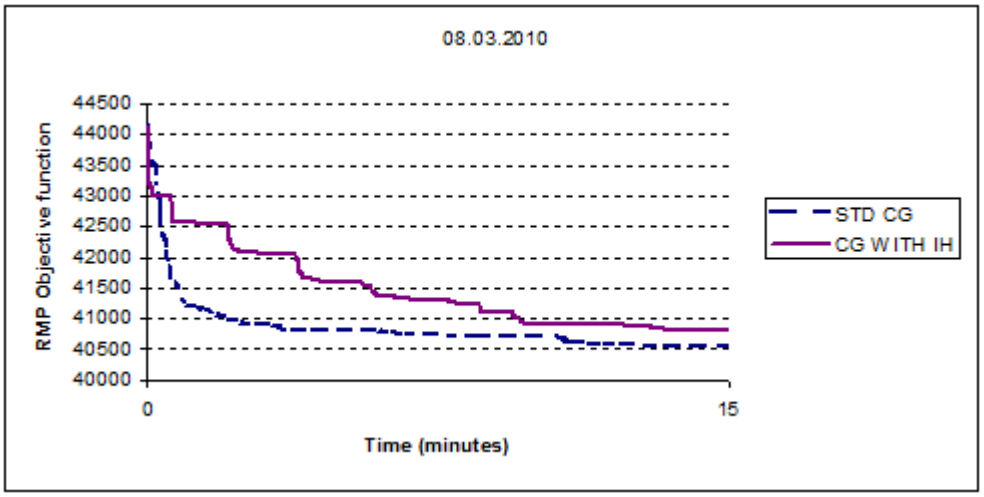


Figure 5.1: The objective function value of the relaxation of the RMP during the resolution of the instance 08.03.2010 (other instances have the same trend)

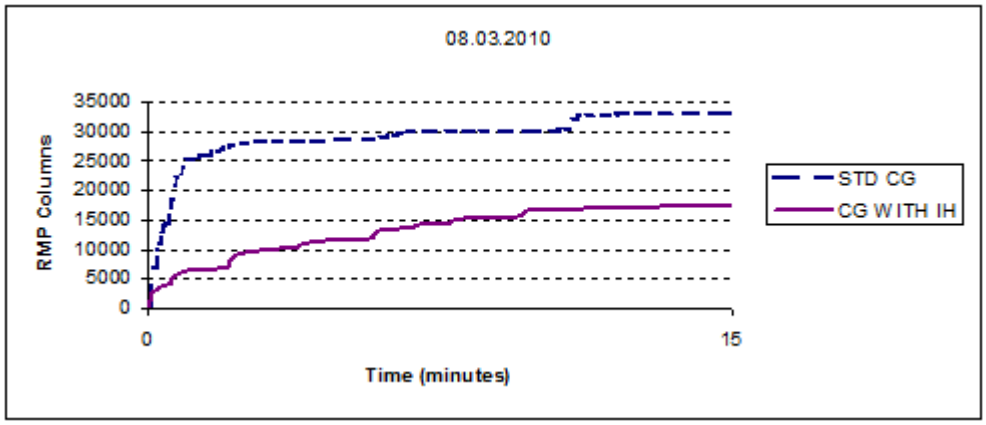


Figure 5.2: The number of columns within the RMP's relaxation during the resolution of the instance 08.03.2010 (other instances have the same trend)

	STD CG					CG WITH IH				
Setting	cost €	cols	RMP	mip*	best (%)	cost €	cols	RMP	mip*	best (%)
s_30_0_0	31185.19	-	-	-	7	31185.19	-	-	-	7
s_10_5_15	30348.30	44744	28761.38	2	7	30270.87	12914	29461.15	12	0
s_10_10_10	30394.16	47864	28629.83	0	14	**29930.16**	16898	28971.72	5	29
s_10_15_5	30646.28	49022	28594.71	0	7	29954.75	18763	28821.76	3	29
s_0_20_10	-	53255	-	2	0	-	29713	-	5	33

Table 5.9: Aggregated results

Chapter 6

Dealing with real world VRPs

Modelling a vehicle routing problem for a real world logistic company means reformulate the set of rules used by the planners into a mathematical model. Those rules prove the feasibility of a tour and its reformulation is a complex task because they often encode empirical knowledge. Modelling activities affect the execution time of the algorithm and the quality of the provided solution (whatever optimization technique is used!), as consequence they are a key factor for the satisfaction of the customer.

Once the problem has been modeled, implemented and tuned we wanted to evaluate its performances.

6.1 Evaluating the model

The proposed evaluation has two goals:

1. to evaluate the usefulness of the results for our commerical customer (this analisys measures how the solution is good in a real world environment)

2. to compare the cost of the solution computed by the algorithm within a reasonable time-limit with a good lower bound.

In our case, the evaulation is based on the comparaison between two hand made plans with the solutions of the algorithm. Our commerical customer gave us two hand made plans concerning the 06.08.2010 and the 23.08.2010 (to manually define a plan require 20 man-hours: 5 employees for 4 hours). For our experiments we executed the algorithm within a timelimit of 30 minutes (a reasonable amount of time in according with the customer requirement).

In Table 6.1 we report the results of the comparaison. The first column reports the name of the instance, in our case the day of the plan. The next

column reports the cost of the plans done by hand. Then we report the cost and the gap of the plans computed within 30 minutes of algorithm execution. The gap was computed using the LB which is the next column. The LB was computed using the RMP value once the root node is exactly solved. The last column reports the cost of the best known integer solution which was computed solving the RMP, once the root node is exactly solved, with the integrality constraint on the tours.

From the examination of Table 6.1 it is possible to draw some conclusion:

- the algorithm saves money, respectively 598 and 590 euro.
- the algorithm saves time (30 minutes instead of 20 man-hours).
- the algorithm within a time-out of 30 minutes provide a solution with a gap of respectively 4.35% and 6.34% from the lower bound.

	Real world	Model			
	20 mh plan	30 min. algorithm		Full algorithm	
Instance	cost €	cost €	gap %	LB	Best integer cost €
06.08.2010	16723	16125	4.35	15423	16013
23.08.2010	24163	23573	6.34	22079	23533

Table 6.1: Comparison between the handmade plan and the algorithm solution's

6.2 Evaluating the hand made plan

Based on our analysis we discovered that 90% of the tours of the hand made solution are not feasibles according to the constraints of the model. That means an employee can do something the algorithm can't. As example, we report a missed tratta violation:

Assuming that a vehicle could profitabily delivery some goods in Piemonte but, according with its tariff configuration (see Section 2.3), it can't. A traffic office employee can call the driver of the truck, ask if he can exceptionally drive in Piemonte, ask how much does it would cost and, finally, decide to use that vehicle for the tour. The algorithm, conversely, will simply discard the tour.

Time windows	**3** violations of 52.38, 63.14 and 310.71 minutes.
Driving time	**1** violation of 8.69 minutes.
Working time	**2** violations of 20.61 and 1.69 minutes.
Cost function	**2** violations due to missed tratta.
Compatibility location-vehicle	**1** violation (used vehicle of type MOT where location doesn't allow it).
Max. distances between stops	**1** violation of 63.49 km (delivery-delivery);**2** violations of 3.55 and 501.52 km (delivery-pickup); **1** violation of 24.32 km (pickup-pickup).
Weight and volume	**2** violations of 11.28 and 0.16 pallet; **3** violations of 307.4, 631.68 and 3686.15 kg.
Primary and secondary zones	**1** violation of 11.37 km (against the max. length limit of tour within its zone).

Table 6.2: 19 Violations of the 06.08.2010

Tables 6.2 and 6.3 report the detailed description of each violation discovered in the hand made plans. Table 6.2 reports the 19 violations discovered the 06.08.2010, while Table 6.3 reports the 28 violations of the 23.08.2010. Most of them represent minor violations but in the plan of 06.08.2010 we report a "Weight and volume" violation of 3686.15 kg which represents the 14% of the total capacity of the vehicle!

The high number of violations existing in the hand made plans induced us to perform additional tests. First we analyzed how the constraints were violated, then we defined a set of relaxation to be applied to our model. Table 6.4 reports the datails of the applied relaxation (we noticed that is not possible to apply a generic relaxation to handle the "missed tratta" violation within the "Cost function" constraint). We executed our algorithm on the relaxed model.

In Table 6.5 we report the results of the comparaison between the solutions of the original model and solutions of the relaxed model. The first 2 columns report the cost and the gap of the plan computed by the algorithm within the original model. The gap was computed using the LB which is the next column. The LB was computed using the RMP executed within the original model once the root node is exactly solved. The next column reports the cost of the best known integer solution which was computed solving the RMP within the original model, once the root node is exactly solved, with the integrality constraint on the tours. The next 4 columns have the same content but they were computed

Time windows	**3** violations due to visiting closed locations; **2** violations of 141.4 and 66.38 minutes.
Driving time	**1** violation of 24.89 minutes.
Cost function	**7** violations due to missed tratta.
Compatibility location-vehicle	**2** violations (used vehicle of type MOT, respectively BIL where location doesn't allow its).
Minimum delivery	**1** violation of 234 km.
Max. distances between stops	**2** violations of 11.92 and 0.84 km (delivery-delivery); **3** violations of 8.53, 30.39 and 49.35 km (pickup-pickup).
Weight and volume	**6** violations of 6.9, 12.1, 3.2, 15.4, 6.4 and 2.7 kg.
Primary and secondary zones	**1** violation of 273.64 km (against the max. length limit of tour within its zone).

Table 6.3: 28 Violations of the 23.08.2010

Working time + 3 % (e.g. the time limit was extent from 900 to 927 minutes)
Driving time + 3 % (e.g. the time limit was extent from 630 to 649 minutes)
Maximal distances between deliveries + 15 % (e.g. the maximal distance allowed for two consecutives deliveries was extent from 200 to 230 km)
Maximal distances between pickups + 15 % (e.g. the maximal distance allowed for two consecutives pickups was extent from 110 to 127 km)
Maximal distances between delivery-first pickup + 15 % (e.g. the maximal distance allowed was extent from 90 to 104 km)

Table 6.4: Applied relaxations

within the model relaxed.

From the examination of Table 6.5 it is possible to draw some conclusion:

- as expected, a relaxation of the model allows to improve the lower bound (-244 euro the 06.08.2010 and -270.8 euro the 23.08.2010) and the cost of the best integer solution (-270.1 euro the 06.08.2010 and -513.1 euro the 23.08.2010).
- within a 30 minutes time-limit setting, a relaxation doesn't helps necessarily the algorithm (+58 euro the 06.08.2010 and -543 euro the 23.08.2010). This is because the dimensions of the search space increase and the algorithm can not explore it properly within 30 minutes.

Instance	Model				Model relaxed			
	30 min. algorithm		Full algorithm		30 min. algorithm		Full algorithm	
	cost €	gap %	LB	Best integer cost €	cost €	gap %	LB	Best integer cost €
06.08.2010	16125.2	4.55	15423.0	16013.2	16183.2	6.61	15178.6	15743.1
23.08.2010	23572.7	6.76	22079.0	23532.8	23029.8	5.60	21808.2	23019.7

Table 6.5: Comparison between the results computed within the original model and those computed in the relaxed model

6.3 Conclusions

From the examination of the chapter it is possible draw some conclusions about our experience applying an optimization algorithm to solve a real world problem.

The algorithm doesn't allow to obtain a huge improvement of the hand made solution (Table 6.1 shows an improvement of 3 %) but it is faster (30 minutes against 20 man-hours).

Customers define constraints which are often violated in the real life; conversely, the solution provided of the algorithm respects all those constraints.

Chapter 7

Conclusions

Vehicles Routing Problem is a difficult combinatorial optimization problem for which different types of algorithm are available in literature. We provided a column generation based heuristic for a Real-World VRP. The algorithm is effective and usable in practice. Moreover, we devised a non trivial and effective incremental heuristic to speedup the process. The proposed framework is a generic scheme to solve large-scale routing problems and provides good quality solutions.

7.1 Future Work

The model presented in the thesis can be extended to cope with additional features (e.g., dynamic customer aggregation, soft constraints). Another possible extensions is to consider the use of stochastic data instead of deterministic data.

Appendix A

Summary of scientific articles

The analisys of the problem and its implementation were preceded by the revision of the most important articles concerning both the column generation principle and its application to solve rich vehicle routing problem. Each article has been summarized and reported as a paragraph whithin this chapter.

Branch and Price for the Vehicle Routing Problem with Discrete Split Deliveries and Time Windows The paper (Salani and Vacca, 2009) describes the resolution of a Discrete Split Delivery Vehicle Routing Problem with Time Window (DSDVRPTW) via branch-and-price. Deliveries can be discreetly splitted in facts the problem provides different orders which are combination of items to be delivered. The algorithm will select the more advantageous combination (e.g. is better delivery items within a single customer order {a,b,c,d} or plan two different tours delivering separately the available splitted subset {a,b} and {c,d} ?).

The problem has been first modeled as a mixed integer problem based on an arc-flow formulation (this MIP formulation is not polynomlially solvable), then it has been reformulated via Dantzing-Wolfe to obtain the restricted master problem (RMP) which has been solved using column generation. The pricing subproblem as been modeled as an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) defined on a network which has a node for every order. The resulting search tree is explored using a best-first strategy.

The algorithm has been tested on instances up to 50 customers with a time limit of 1 hour. Remarkable, the splitting property of the problem requires much more computational time but it provides better solution because the service time is proportional with the size of order, this feature let be more promising the use of smaller order.

Selected topic in column generation The paper (Lubbecke and Desrosiers, 2005) merges promising contemporary research works with more classical solution strategies to cover the whole integer programming column generation process.

The paper introduce the theory of column generation: *Let us call the following linear program the master problem (MP):*

$$\begin{aligned} z^* = min & \sum_{j \in J} c_j \lambda_j \\ subject\ to & \sum_{j \in J} c_j \lambda_j \geq b, \\ & \lambda_j \geq 0, j \in J \end{aligned} \tag{1}$$

In each iteration of the simplex method we look for a nonbasic variable to price out and enter the basis. That is, in the pricing step, given the vector $u \geq 0$ of dual variables, we wish to find

$$arg\ min\{\bar{c}_j := c_j - u^T a_j \mid j \in J\}. \tag{2}$$

An explicit search of J may be computationally impossible when $|J|$ is huge. In practice, one works with a reasonably small subset $J' \subseteq J$ of columns, with a restricted master problem (RMP). Assuming that we have a feasible solution, let $\overline{\lambda}$ and $\overline{u}$ be primal and dual optimal solutions of the RMP, respectively. When columns a_j, $j \in J$, are implicitly given as elements of a set $\mathscr{A} \neq \emptyset$, and the cost coefficient c_j can be computed from a_j , then the subproblem or oracle

$$\overline{c}^* := min\{c(a) - \overline{u}^T a \mid a \in \mathscr{A}\}. \tag{3}$$

returns an answer to the pricing problem. If $\overline{c}^* \geq 0$, no reduced cost coefficient $\bar{c}_j$ is negative and $\overline{\lambda}$ (embedded in $\mathbb{R}^{|J|}$) optimally solves the MP as well. Otherwise, we add to the RMP a column derived from the oracle's answer, and repeat with re-optimizing the RMP. For its role in the algorithm, (3) is also called the column generation subproblem, or the column generator. The advantage of solving an optimization problem in (3) instead of an enumeration in (2) becomes even more apparent when we remember that vectors $a \in \mathscr{A}$ often encode combinatorial objects like paths, sets, or permutations. Then, $\mathscr{A}$ and the interpretation of cost are naturally defined on these structures, and we are provided with valuable information about what possible columns "look like". With the usual precautions against cycling of the simplex method, column generation is finite and exact. In addition, we have a knowledge about the intermediate solution quality during the process. Let $\overline{z}$ denote the optimal objective function value to the RMP. Note that by duality we have $\overline{z} = \overline{u}^T b$. Interestingly, when an upper bound $k \geq \sum_{j \in J} \lambda_j$ holds for an

optimal solution of the master problem, we estabilish not only an upper bound on z^* in each iteration, but also a lower bound: We cannot reduce $\overline{z}$ by more than k times the smallest reduced cost $\overline{c}^*$, hence,

$$\overline{z} + \overline{c}^* k \leq z^* \leq \overline{z}. \tag{5}$$

In the optimum of (1), $\overline{c}^* = 0$ for the basic variables, and the bounds close. The lower bound in (5) is computationally cheap and readily available when (3) is solved to optimality.

The paper provides an overview of the Dantzing and Wolfe decomposition principle in linear programming. This technique allows to transform the original or compact formulation into the extensive formulation which is relaxed in order to obtain the master problem. The trasformation is based on the convexification conv(x) of X. The polyhedron defined by conv(x) is equivalent of that defined by X (the optimal solution z^* is the same), but it has integral vertices. The steps required to obtain the master problem are the following:

1. the original or compact formulation (an integer problem) is transformed into an extensive formulation (a mixed integer linear problem). In the extensive formulation variables x are integrals (thanks to the convexification) and variables λ are relaxed.

2. the extensive formulation without the x variables becomes the master problem which is its linear relaxation (a linear problem). The master problem only deals with λ. λs are the columns of the problem. A λ is a convex combination of extreme points plus a non negative combination of extreme rays of the convexed polyhedra. Add a λ to the problem means add a vertex to the polyhedra. A λ intrinsically encodes the removed x. Since $|\lambda|$ is a huge (exponential) number the master problem is solved using column generation.

Implementing Mixed Integer Column Generation The paper (Vanderbeck, 2005) reports the two types of constraint of mixed integer problems: difficult constraints (e.g. covering constraints in TSP, customer request satisfaction in CUTTING STOCK) or more tractable combinatorial subproblems (e.g. find feasible tours TSP, find feasible pattern in CUTTING STOCK). Such a structure allows to reformulate the problem (via Dantzing-Wolfe) in order to obtain a master problem (which manages difficult constraints) and a pricing subproblem (which handles more tractable constraints). The master will be solved optimally via column generation.

The paper reports strategies to solve the master problem. The master can be solved using the revised simplex algorithm (dual variables obtained from the

resolution of the master are used to drive the column generation) but also using other methods (e.g. the bundle method and ACCPM) to refine the dual solution allowing a more stable convergence.

The paper explains that there are many way to set up the Dantzing-Wolfe reformulation of a MIP (choose if constraints are putted in the master or in the pricing, model the pricing using dynamic programming, aggregate variables,...). Basically the trade-off is between have good dual bounds and have problems (master and pricing) which can be solved fast.

The main decisions to set up a branch and price algorithm (after master and pricing problem are defined) are 1) which rule follow to discretize non-integral columns (branching scheme) 2) how choose components on which to branch? 3) how explore the search tree (best bound first, depth first,...) The two main frameworks where implement such decisions are discretization and convexification. Those decisions depends on the type of the problem and can have a big impact on the convergence ability of the method.

The paper reports some preprocessing which can be applied in the original formulation solution space (which is also the space of the subproblem variables) in order to obtain tighter bounds allowing the elimination of redundant constraints or proving infeasibility. Preprocessing rely on generate more proper column. If an incumbent integer solution INC is available it can be used to dominate solution where a variable takes value within restricted range.

The paper reports some initialization techniques for the master. The simplex algorithm used to solve the column generation formulation must be started (in the root node) with a feasible primal solution which can be constructed heuristically or introducing artificial columns (they can be reused during the branching procedure, their cost is increased to remove them from the solution). The reinitialization of nodes other then the root can be done using the already generated columns (taken from the parent node) which satisfy the branching constraint and with a zero reduced cost. An intelligent initialization of the restricted master problem offer a warm start.

The paper reports known drawbacks of the simplex algorithm with dynamic pricing of variables i) slow convergence ("tailing-off effect"), ii) first iteration produce irrelevant columns ("heading-in effect"), iii) degeneracy in primal solution (the objective function doesn't improve since the simplex is exploring a face of the polyhedron) and multiple optimal solution in the dual ("plateau effect"), iv) dual solution that are jumping from one extreme value to another ("bang-bang effect").

The paper reports some heuristic strategies which can be used within a column generation algorithm: it is possible re-optimize existing columns (the optimization is done after updating dual price with current columns and before

re-call the oracle to generate new columns) and apply a post-processing to sub-problem solutions to increase column coefficients in master constraint. It is also possible use heuristic strategies to initialize the primal solution (e.g. to do a greedy genaration of columns with smallest ratio of cost per unit of constraint satisfaction and to implement a local search to exchange columns in a restricted pool).

A Column Generation Algorithm for a Rich Vehicle-Routing Problem The paper (Ceselli et al., 2009) presents an optimization algorithm which computes a daily plan for a rich vehicle-routing problem managing: (i) heterogeneous fleet of vehicles (ii) departure from different depots (iii) multiple capacities vehicles (iv) time windows associated with depots and customers (v) incompatibility constraints between goods, depots, vehicles, and customers (vi) maximum route length and duration (vii) upper limits on the number of consecutive driving hours and compulsory drivers' rest periods (viii) the possibility of skipping some customers and using express courier services instead of the given fleet to fulfill some orders (ix) the option of splitting up the orders (x) the possibility of open routes that do not terminate at depots (xi) the cost of each vehicle route is computed through a system of fees. That algorithm, which is based on column generation and dynamic programming, is able to compute optimal solutions or near-optimal ones with a posteriori approximation guarantee.

The paper provides a description of the problem. There are two types of locations (depots and customers) which are identified within a hierarchical structure. For each couple of locations we know the distance (time and space). For each location we know: time windows, load and unload unit service time, set of mandatory orders, max number of visit per day. An order is made of several items. An item has a weight, a volume and a value. Orders can be splitted but item cannot. A vehicle has a daily duty composed of many routes each starting at the same depot. Vehicles have a type describing max weight, max volume, max value, max number of pallet, max daily duty length (time and space), max length per root, max stop per route. Unload time is a weighted sum of weight, volume, value and number of pallets. Load time is a constant which depends on the vehicle type. Incompatibilities exists between items-depots, items-customers, vehicle types-items and items-items. Routes have a duration which is the sum of traveling times, waiting times, loading unloading times. Traveling time must take into account the driver's rest period. The cost function is computed through a hierarchical fees structure (based on nation, zone, region, district and zip code). More characteristics of the tour are taken into account to compute the cost (travel distance, number of pallet, weight, volume, value, stops).

The paper provides the mathematical formulation of the master problem.

This problem is a set-covering problem (SCP) in which each column encodes a feasible duty (|columns| is exponential). That problem has been modeled as an integer linear problem which select the cheapest duties (in according with the cost provided by the pricing) managing following constraints: (i) each item is delivered, (ii) vehicle type availability constraint (the number of duties selected for a given type of vehicle can not exceed the number of available vehicle of that type), (iii) max number of visit per day in locations. Relaxing integrality in SCP we obtain a restricted set-covering problem (RSCP) which is used as master problem. The master problem will work using a subset of columns in a column generation algorithm. We observe that "globals" constraints (related to a daily plan level) are managed in the master.

The paper describes the pricing problem. The pricing problem is a special case of the resource-constrained elementary shortest-path problem (RCESPP) formulated on a graph having one vertex for each item, nonnegative costs on the arcs, nonnegative prizes on the vertices and nonnegative penalties for entering each location (costs are the computed using fees system, prizes and penalties are given by the dual variables). RCESPP is NP-hard and it is solved optimally using the bounded bidirectional dynamic programming algorithm (BBDP) (Righini and Salani, 2006). When considering a vehicle of type p and a depot of type d we discard items that are either unavailable at depot d or incompatible with vehicle type p. We observe that "locals" constraints (related to a duty level e.g. time windows, max vehicle load,...) are managed in the pricing.

The authors of the paper affirm that the harder requirement encountered in the optimization algorithm is the splittability of the order. For this reason the algorithm has been devided in three phases. The first phase produce very quickly a feasible solution, the second and the third phase introduce gradually the splittabily of the order and try to find improving column exploiting that feature.

Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem The paper (Dumitrescu and Boland, 2003) presents a computational comparison of three scaling techniques and a standard label-setting method in order to solve a Weight Constrained Shortest Path Problem (WCSPP). The WCSPP is an NP-hard problem which consists of finding a minimum cost weight feasible path P in a directed graph G from s to t (P minimizes cost $c(P)$ and satisfies constraints $w(P) \leq W$). The comparison has been done using different datasets (obtained through various techniques e.g. random generation of costs and weight for each arc).

The authors propose a preprosessing algorithm for a RCSPP (which is a WCSPP with a vector of weights for each arc) network. That algorithm has been

compared with similar methods. The reported results highlight how a proper preprocessing techniques allows to solve the problem or to reduce (through the removal of nodes and arcs) them to an almost trivial size. Remarkable: the algorithm allows to detect the feasibility of the problem. The performance of the algorithms are affected by factor such as the strength of the weight limits (e.g. some algorithms work well with medium and high-weight limit problems but not with low-weight limit problem).

The authors describe the principles of the LSA and presents its modified version of that algorithm called MLSA. The proposed algorithm integrates information obtained in preprocessing. The LSA and the MLSA are been compared and the MLSA results markedly better than does the LSA (it can solve more problem using less memory and often more fastly).

The authors consider the fully polynomial approximation schemes (FPASs) which is an algorithm solvable in polynomial time thanks to the scaling of costs. The FPASs can be seen to be equivalent to the label-setting algorithm (LSA). The FPASs principles are been exploited in order to obtain two algorithms: H-FPAS (which doesn't use preprocessing information) and L&R-FPAS (which uses preprocessing information). The H-FPAS and the L&R-FPAS are been compared and the results highlights how the performances depend on the problem itself, in facts not always the use of preprocessing informations together with a scaling technique is convenient. Morover, the authors demonstrate how the approximation algorithms suffer from a complexity dependent on the magnitude of the costs and that the label-setting algorithms while coping with costs of large magnitude without affecting their complexity, do not provide lower bounds.

The authors propose an exact algorithm (W-S) which scales weight and then uses preprocessing information in a reduced MLSA (LSA with preprocessing informations). This algorithm allows to handle costs of large magnitude and provide good lower bounds in modest time is to use an based on weight scaling. The algorithm has been tested and compared with others proposed algorithms: in a problem class it is resulted as the best.

The paper reports a numerical comparaison of the five presented algorithms able to solve the WCSPP (three exact algorithms: LSA, MLSA, W-S; and two approximation algorithms: H-FPAS and L&R-FPAS). The best is resulted the MLSA (use preprocessing informations in LSA is very useful).

The paper reports a Lagrangean relaxation approach which can be used to obtain stronger lower bounds to be used to prune labels in a LSA. It is also possible make use of information collected in the solution of the Lagrangean dual to improve upper bounds. This startegy used in a MLSA allowed to solve faster and using less memory the considered problems.

The conclusion of the paper is that a simple preprocessing techniques can be

surprisingly effective.

Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints The paper (Righini and Salani, 2006) considers dynamic programming algorithms for the resource constrained elementary shortest path problem (RCESPP). The RCESPP is the problem of finding the minimum cost elementary path (path in which a node is visited at most once so that the path does not contain a cycle) from a node s to a node t of a given graph such that the overall amounts of resources consumed do not exceed some given limits. The vertices of the graph are customers (s and t are two special vertices representing the depot). A non-negative integer cost is associated with each arc. A non-negative prize is associated with each vertex and a non-negative cost is associated with the depot. A vehicle must go from s to t, visiting a subset of the other vertices. The objective is to minimize the cost, given by the sum of the costs of the arcs traversed minus the sum of the prizes collected at the vertices visited. When the underlying graph have negative cost cycles, the resource constrained elementary shortest path problem is strongly NP-hard (Dror, 1994).

The paper provides a description of the dynamic programming algorithm for the RCESPP which is is based on the algorithm devised by (Desrosiers et al., 1981) for the RCSPP which is an extension of the Bellman-Ford algorithm with the addition of resource constraints. The resource constraints are of many kind: capacity (depends on vertices visited along a path), distribution and collection (both depend on vertices visited and they are interacting in facts the amount of doable load depends on the amount of doable pickup and vice versa), capacity and time windows (one depends on vertices visited and the other associated with the arcs traversed). The dominance tests ensure the effectiveness of the dynamic programming algorithm. Dominance tests allows to extends only promising state. Extended states are not deleted, because they can be useful to dominate other states not yet generated.

The paper reports a mono-directional algorithm to solve the RCESPP. That algorithm generates a number of labels which, in the worst case, grows exponentially with the number of arcs in the path moreover only dominated states can be fathomed. In order to mitigate the weaknesses of the mono-directional algorithm it is proposed a bi-directional version which considers two smaller paths, a backward and a forward, instead of a single path. Since the complexity of the algorithm depends on the length of the path, this approach (together with a bounding mechanism) allows to speed up the performances of the algorithm. The proposed bi-directional approach is feasible thanks to the symmetry of the resources, between the backward and forward path, which allows them to be

joined. The dyamic programming algorithm generates a set of states which can be explored in according with different search strategies (e.g. label-correcting which visits cyclically vertices to extends states; label-setting which requires a monotonicity consumption of resources). The proposed bi-directional algorithm applies (on both directions) a label-correcting algorithm over states ordered by non-decreasing resource consumption. Bounding is used to fathom unpromising states (states are evaluated end compared with the UB given by a known feasible solution to decide if they must be fathom) and to stop their extension (evaluation of number of arcs that can be added to the path and of critical resources consumption). The strategy of the algorithm (to do backward and forward paths for states) results on the generation of duplicated solution (more combination of states representing the same tour).The algorithm implements mechanisms to detect such a combination avoiding them.

The mono-directional algorithm and the bi-directional algorithm (with arc bounding and with resource bounding) are been compared using five datasets (RCESPP with capacity - 100 vertices; ESPPRC with capacity and time window - 50 and 100 vertices; RCESPP with capacity and time window - 50 and 100 vertices). The resource bounded bi-directional algorithm solves more and larger instances than the mono-directional algorithm and it reduces the computing time by one order of magnitude. Interesting to note how the time window setting can affect the execution time of the bi-directional algorithm, in facts when time windows become larger and larger, the number of non-dominated states increases dramatically.

Appendix B

A RMP practical execution example

In this section we propose an example which shows how the RMP model "appears" and how it changes during the column generation execution's. Iteration by iteration we monitor the dual variables changes and we explain how the objective function of the pricing would react to those changes.

Iteration 0 Table B.1 reports a typical master problem just after its initialization. The model is a set covering where the elements to be covered are the items of the problem, the sets are the tours and we have an additional constraint to manage the availability of vehicle types. In our example we have 3 items (y_i1,y_i2, y_i3), 2 vehicle types (1 vehicle of type v_1, 3 vehicles of type v_2) , 3 rows to manage the covering constraint (cov_i1, cov_i2, cov_i3), 2 rows to manage the vehicle availability constraint (ava_v1, ava_v2).

Table B.2 reports the dual values of the constraint once model B.1 is solved. For the objective function of the pricing all the items and all the vehicles have the same prizes since $\gamma_1 = \gamma_2$ and $\pi_1 = \pi_2 = \pi_3$.

$$
\begin{aligned}
&Minimize \\
&\quad obj: +10000y\_i1 + 10000y\_i2 + 10000y\_i3 \\
&SubjectTo \\
&\quad cov\_i1: +y\_i1 >= 1 && (\pi_1) \\
&\quad cov\_i2: +y\_i2 >= 1 && (\pi_2) \\
&\quad cov\_i3: +y\_i3 >= 1 && (\pi_3) \\
&\quad ava\_v1: +dummy\_v1 <= 1 && (\gamma_1) \\
&\quad ava\_v2: +dummy\_v2 <= 3 && (\gamma_2) \\
&Bounds \\
&\quad 0 <= y\_i1 <= 1 \\
&\quad 0 <= y\_i2 <= 1 \\
&\quad 0 <= y\_i3 <= 1 \\
&\quad dummy\_v1 = 0 \\
&\quad dummy\_v2 = 0 \\
&End
\end{aligned}
$$

Table B.1: Iteration 0 RMP

$$
\begin{aligned}
obj &= 30000 \\
y\_i1 = y\_i2 &= y\_i3 = 1 \\
\pi_1 = \pi_2 &= \pi_3 = 10000 \\
\gamma_1 &= \gamma_2 = 0
\end{aligned}
$$

Table B.2: Iteration 0 duals

Iteration 1 we are adding $\boldsymbol{kv1\_0}$ (a column which costs 10, of vehicle $v1$, covering item $i1$ and $i2$). Table B.3 reports the updated model. Table B.4 reports the dual values of the constraints once the model B.3 is solved. In according with Table B.4, for the objective function of the pricing all the items and all the vehicles have the same prizes since $\gamma_1 = \gamma_2$ and $\pi_1 = \pi_2 = \pi_3$.

$$
\begin{aligned}
&Minimize \\
&\quad obj: +10000y\_i1 + 10000y\_i2 + 10000y\_i3 + \mathbf{10kv1\_0} \\
&SubjectTo \\
&\quad cov\_i1: +y\_i1 + \mathbf{kv1\_0} >= 1 && (\pi_1) \\
&\quad cov\_i2: +y\_i2 + \mathbf{kv1\_0} >= 1 && (\pi_2) \\
&\quad cov\_i3: +y\_i3 >= 1 && (\pi_3) \\
&\quad ava\_v1: +dummy\_v1 + \mathbf{kv1\_0} <= 1 && (\gamma_1) \\
&\quad ava\_v2: +dummy\_v2 <= 3 && (\gamma_2) \\
&Bounds \\
&\quad 0 <= y\_i1 <= 1 \\
&\quad 0 <= y\_i2 <= 1 \\
&\quad 0 <= y\_i3 <= 1 \\
&\quad dummy\_v1 = 0 \\
&\quad dummy\_v2 = 0 \\
&\quad \mathbf{0 <= kv1\_0 <= 1} \\
&End
\end{aligned}
$$

Table B.3: Iteration 1 RMP

$$
\begin{aligned}
obj &= 10010 \\
kv1\_0 &= 1, y\_i3 = 1 \\
\pi_1 &= \pi_2 = \pi_3 = 10000 \\
\gamma_1 &= \gamma_2 = 0
\end{aligned}
$$

Table B.4: Iteration 1 duals

Iteration 2 we are adding $\boldsymbol{kv1\_1}$ (a column which costs 8, of vehicle $v1$, covering item $i3$). Table B.5 reports the model once the column $kv1\_1$ was added. Table B.6 reports the dual values of the constraints once the model B.5 is solved. In according with Table B.6, the objective function of the pricing would prefer to generate columns for vehicle v_2 which has associtaed the bigger dual prize, in facts $\gamma_2 > \gamma_1$.

$$
\begin{aligned}
&Minimize \\
&\quad obj: +10000y\_i1 + 10000y\_i2 + 10000y\_i3 + 10kv1\_0 + \mathbf{8kv1\_1} \\
&SubjectTo \\
&\quad cov\_i1: +y\_i1 + kv1\_0 >= 1 && (\pi_1) \\
&\quad cov\_i2: +y\_i2 + kv1\_0 >= 1 && (\pi_2) \\
&\quad cov\_i3: +y\_i3 + \mathbf{kv1\_1} >= \mathbf{1} && (\pi_3) \\
&\quad ava\_v1: +dummy\_v1 + kv1\_0 + kv1\_1 <= 1 && (\gamma_1) \\
&\quad ava\_v2: +dummy\_v2 <= 3 && (\gamma_2) \\
&Bounds \\
&\quad 0 <= y\_i1 <= 1 \\
&\quad 0 <= y\_i2 <= 1 \\
&\quad 0 <= y\_i3 <= 1 \\
&\quad dummy\_v1 = 0 \\
&\quad dummy\_v2 = 0 \\
&\quad 0 <= kv1\_0 <= 1 \\
&\quad \mathbf{0 <= kv1\_1 <= 1} \\
&End
\end{aligned}
$$

Table B.5: Iteration 2 RMP

$$
\begin{aligned}
obj &= 10010 \\
kv1\_0 = y\_i3 &= 1 \\
\pi_1 = \pi_2 = \pi_3 &= 10000 \\
\gamma_1 = -9992, \gamma_2 &= 0
\end{aligned}
$$

Table B.6: Iteration 2 duals

Iteration 3 we are adding $\boldsymbol{kv2\_0}$ (a column which costs 5 , of vehicle $v2$,covering items $i1$ and $i2$). Table reports B.7 the updated model. Table B.8 reports the dual values of the constraints once the model B.7 is solved. In according with Table B.8, the objective function of the pricing would prefer to generate columns for vehicle v_2 covering items i_1 and i_3 which have the bigger dual prizes, in facts $\gamma_2 > \gamma_1$ and $\pi_2, \pi_3 > \pi_1$.

$$
\begin{aligned}
&Minimize \\
&\quad obj: +10000y\_i1 + 10000y\_i2 + 10000y\_i3 + 10kv1\_0 + 8kv1\_1 + \mathbf{5kv2\_0} \\
&SubjectTo \\
&\quad cov\_i1: +y\_i1 + kv1\_0 + \mathbf{kv2\_0} >= 1 && (\pi_1) \\
&\quad cov\_i2: +y\_i2 + kv1\_0 + \mathbf{kv2\_0} >= 1 && (\pi_2) \\
&\quad cov\_i3: +y\_i3 + kv1\_1 >= 1 && (\pi_3) \\
&\quad ava\_v1: +dummy\_v1 + kv1\_0 + kv1\_1 <= 1 && (\gamma_1) \\
&\quad ava\_v2: +dummy\_v2 + \mathbf{kv2\_0} <= 3 && (\gamma_2) \\
&Bounds \\
&\quad 0 <= y\_i1 <= 1 \\
&\quad 0 <= y\_i2 <= 1 \\
&\quad 0 <= y\_i3 <= 1 \\
&\quad dummy\_v1 = 0 \\
&\quad dummy\_v2 = 0 \\
&\quad 0 <= kv1\_0 <= 1 \\
&\quad 0 <= kv1\_1 <= 1 \\
&\quad \mathbf{0 <= kv2\_0 <= 1} \\
&End
\end{aligned}
$$

Table B.7: Iteration 3 RMP

$$
\begin{aligned}
obj &= 13 \\
kv1\_1 &= kv2\_0 = 1 \\
\pi_1 &= \pi_3 = 10000, \pi_2 = 0 \\
\gamma_1 &= -9992, \gamma_2 = 0
\end{aligned}
$$

Table B.8: Iteration 3 duals

Appendix C

The software architecture

The columns generation theory has been engineered in order to obtain a framework of classes. The framework has been implemented using Microsoft Visual Studio 2005 and C#.

The result is not only an implemention targeted to solve specifics needs of the commercial partner but also a set of abstract, high level and reusable elements (interconnected interfaces and abstract classes which can be easily extended).

In this section we will explain which are the input, the output, the tasks and the relationships between the elements composing the framework.

The main interfaces of a column generation framework are three IMaster, IPricing and IColumnGenCycle. These three components works togheter in order generate IColumn which are tours done by IVehicle covering some IItem.

IItem This interface represents a node of the trasportation network. It is an item to be covered of the set covering problem. Listings C.1 reports the code of the interface. Method GetLocation (line 2) returns an instance of ILocation which is the geographical location of the item on the transportation network. Method GetDual (line 3) returns the dual prize associated with the item. Method GetItemType (line 4) returns the IItemType which identifies the serivice types (delivery, pickup or pu-and-del). Method getPallet (line 5) returns the quantity of good in pallet represented by the item. Method GetWeightKg (line 6) returns the quantity of good in kg represented by the item.

Listing C.1: interface IItem

```
1 interface IItem {
2     ILocation GetLocation()
3     decimal GetDual();
4     ItemType GetItemType()
5     decimal GetPallet();
6     decimal GetWeightKg();
```

```
7 }
```

IVehicle This interface represents a vehicle types. Listings C.2 reports the code of the interface. Method GetDual (line 2) returns the dual prize associated with the vehicle type. Method GetPallet (line 3) returns the capability of the vehicle type in pallet. Method GetWeightKg (line 4) returns the capability of the vehicle type in kg.

Listing C.2: interface IVehicle

```
1 interface IVehicle {
2     decimal GetDual();
3     decimal GetPallet()
4     decimal GetWeightKg();
5 }
```

IColumn This interface represents a column of the master problem which is a tour performed by a vehicle type. Listings C.2 reports the code of the interface. Method GetVehicle (line 2) returns the vehicle types which performs the tour. Method GetItems (line 3) returns the ordered sequence of items served by the tour. Method GetObjFunctCoeff (line 4) returns the cost of the column (how much does it cost to serve the items using the vehicle type).

Listing C.3: interface IColumn

```
1 interface IColumn {
2     IVehicle GetVehicle();
3     List<IItem> GetItems();
4     decimal GetObjFunctCoeff();
5 }
```

IMaster This interface represents a linear program which can be modified (adding columns), solved and its solution can be inspected. Listings C.4 reports the code of the interface. Method Initialize (line 1) initializes the linear model using the items and the vehicles of the problem. Method AddColumn (line 2) adds the column passed as parameter in the linear model. Method Optimize (line 3) optimizes the model and updates the duals associated with vehicles and items.

Listing C.4: interface IMaster

```
1 interface IMaster {
2     void Initialize(List<IItem> items, List<IVehicle> vehicles);
3     void AddColumn(IColumn columntoadd);
```

```
4     void Optimize();
5 }
```

IPricing This interface represents an optimization algorithm which provides minimal reduced cost columns in according with the dual values encoded in the vehicle and in the items. Listings C.5 reports the code of the interface. Method Initialize (line 2) initializes the pricing with the dual values of the vehicle and of the items. Method Execute (line 3) executes the algorithm. Method GetProfitableColumns (line 4) returns a list of reduced cost columns for the vehicle type, if no reduced cost columns are founded it returns an empty list.

Listing C.5: interface IPricing

```
1 interface IPricing {
2     void Initialize(IVehicle vehicle, List<IItem> items);
3     void Execute();
4     List<IColumn> GetProfitableColumns();
5 }
```

IColumnGenCycle This interface represents the main cycle of the column generation algorithm. It iteratively calls pricings and master until reduced cost columns are founded. Listings C.6 reports the code of the interface. Method Initialize (line 2) initializes the column generation cycle with the problem to be solved (vehicles and items), the master and some pricings (since we used a multi-pricing approach). Method Execute (line 8) executes the column generation cycle.

Listing C.6: interface IColumnGenCycle

```
1 interface IColumnGenCycle {
2     void Initialize(List<IVehicle> vehicles,
3                     List<IItem> items,
4                     IMaster master,
5                     IPricing pricing_greedy,
6                     IPricing pricing_heur,
7                     IPricing pricing_exact);
8     int Execute();
9 }
```

Figure C.1 shows the high level interactions between the interface IColumnGenCycle, IMaster and IPricing. The column generation cycle iteratively 1) solve the master 2) call the pricing which uses the updated duals of the master (right arrow) to find columns with negative reduced cost 3) if the pricing doesn't find

columns with negative reduced cost then the algorithm converges otherwise the found columns (left arrow) are added to the master and the process iterates.

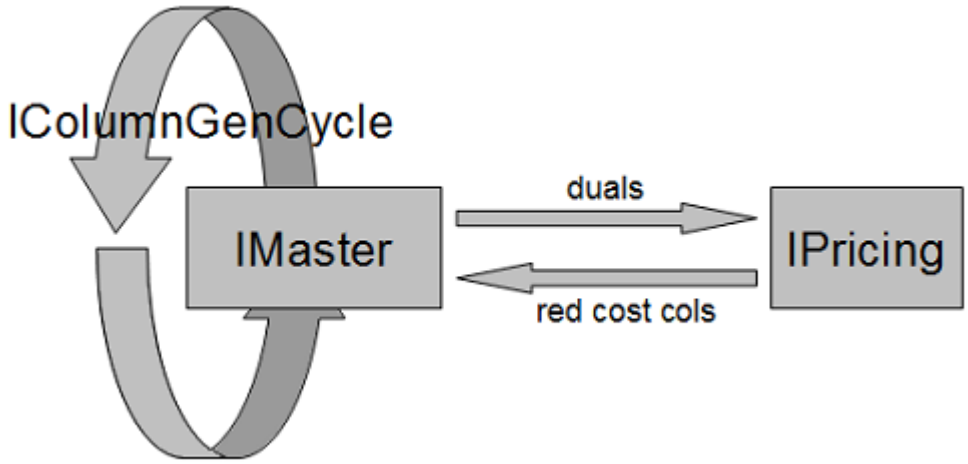


Figure C.1: High level interfaces interactions

Bibliography

Claudia Archetti, Martin W. P. Savelsbergh, and M. Grazia Speranza. Worst-case analysis for split delivery vehicle routing problems. *Transportation Science*, 40: 226–234, May 2006.

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

Alberto Ceselli, G. Righini, and Matteo Salani. A column generation algorithm for a vehicle routing problem with economies of scale and additional constraints. *Transportation Science*, 43(1):56–69, 2009.

G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, 1959.

George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 1960.

G. Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. 58:179 – 192, January-February 2010.

Jacques Desrosiers, Paul Pelletier, and Francois Soumis. *Plus court chemin avec contraintes d'horaires*. 1981.

M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

Moshe Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):pp. 977–978, 1994.

I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3): 135–153, 2003.

Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. *MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows*, pages 63–76. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, first edition edition, January 1979.

Bruce Golden, S. Raghavan, and Edward A. Wasil. *The vehicle routing problem : latest advances and new challenges*. Operations research/Computer science interfaces series, 43. Springer, 2008.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, 1975.

Marco E. Lubbecke and Jacques Desrosiers. Selected topics in column generation. *OPERATIONS RESEARCH*, 53(6):1007–1023, November 2005.

Maciek Nowak, Ozlem Ergun, and Chelsea C. White III. An empirical study on the benefit of split loads with the pickup and delivery problem. *European Journal of Operational Research*, 198(3):734 – 740, 2009.

G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, September 2006.

Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Netw.*, 51: 155–170, May 2008.

A. Rizzoli, R. Montemanni, E. Lucibello, and L. Gambardella. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1(2): 135–151, December 2007.

Matteo Salani and Ilaria Vacca. Branch and price for the vehicle routing problem with discrete split deliveries and time windows. Technical Report TRANSP-OR 091224, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, 2009.

Ramesh Sharda, Stefan VoSS, Claudia Archetti, and Maria Grazia Speranza. The split delivery vehicle routing problem: A survey. In Bruce Golden, S. Raghavan, and Edward Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*, pages 103–122. Springer US, 2008.

Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, 2002.

François Vanderbeck. *Implementing Mixed Integer Column Generation*. 2005.

L.A. Wolsey. *Integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1998.