ILP-BASED REFINEMENT HEURISTICS

Matteo Fischetti, DEI, University of Padova

IBM T.J. Watson Research Center,Yorktown Heights, NY, June 2005

MIP solvers for hard optimization problems

- **Mixed-integer linear programming** (MIP) plays a central role in modelling difficult-to-solve (NP-hard) combinatorial problems

- General-purpose (exact) MIP solvers are very sophisticated tools, but in some hard cases they are **not adequate** even after clever tuning

- One is therefore tempted **to quit the MIP framework** and to design ad-hoc heuristics for the specific problem at hand, thus loosing the advantage of working in a generic MIP framework

- As a matter of fact, too often a MIP model is developed only "to better describe the problem" or, in the best case, to compute bounds for **benchmarking** the proposed ad-hoc heuristics

Can we devise an alternative use of a general-purpose MIP solver, e.g., to address important steps in the solution process?

I MIP you

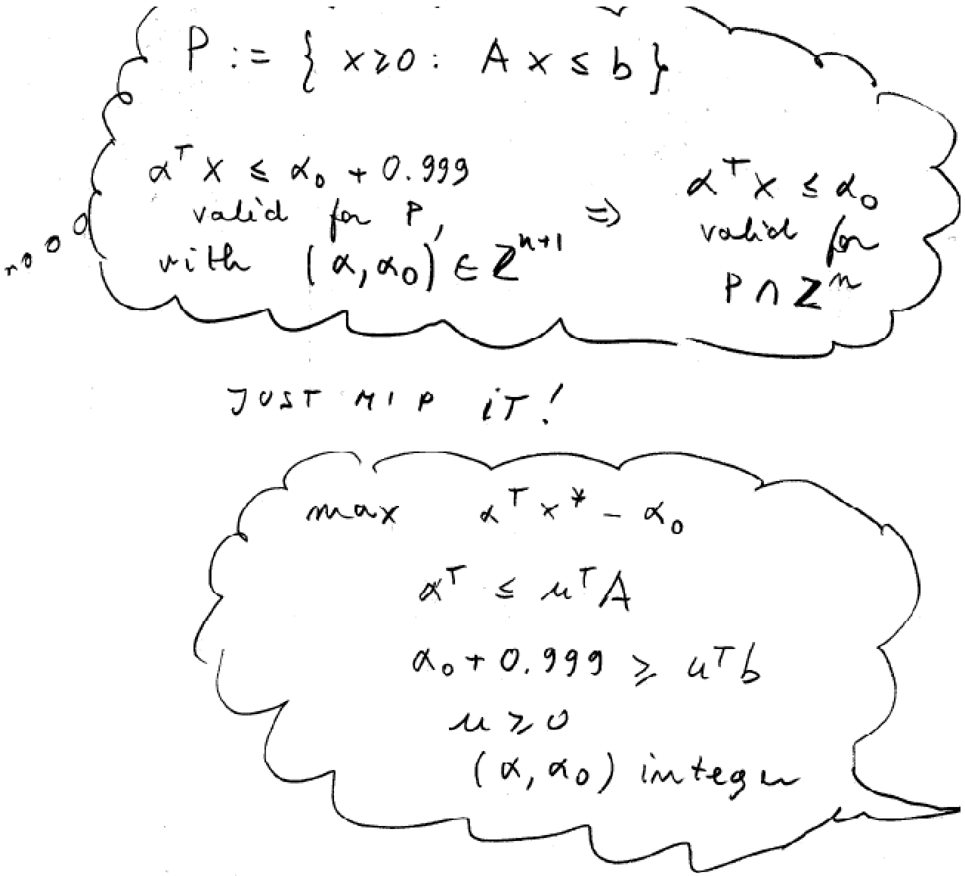
A neologism: To ***MIP something*** = translate into a MIP model and solve through a black-box solver

MIP-heuristic enslaved to an exact MIP solver

- **MIPping Ralph**: use a black-box (general-purpose) MIP heuristic for the separation of Chvàtal-Gomory cuts, so as to enhance the convergence of an exact MIP solver

(M. F., A. Lodi, “Optimizing over the first Chvàtal closure”, IPCO’05, 2005)

MIPped !!!



MIP-solver enslaved to a local-search metaheuristic

MIPping Fred: use a black-box (general-purpose) MIP solver to

- explore large solution neighbourhoods defined through invalid linear inequalities called local branching cuts;
- diversification is also modelled through MIP cuts

(M.F., A. Lodi, “Local Branching”, Mathematical Programming B, 98, 23-47, 2003)

MIPped !!!

Given a feasible 0-1 solution x_H , define a MIP neighbourhood though the **local branching** constraint

$$\Delta(x, x^H) := \sum_{j \in B : x_j^H = 0} x_j + \sum_{j \in B : x_j^H = 1} (1 - x_j) \le k$$

MIPping critical sub-tasks in the design of specific algorithms

We teach engineers to use MIP models for solving **their** difficult problems (telecom, network design, scheduling, etc.)

Be smart as an engineer!

Model the most critical steps in the design of **your own** algorithm through MIP models, and solve them (even heuristically) through a general-purpose MIP solver…

A new heuristic algorithm for the *Vehicle Routing Problem*

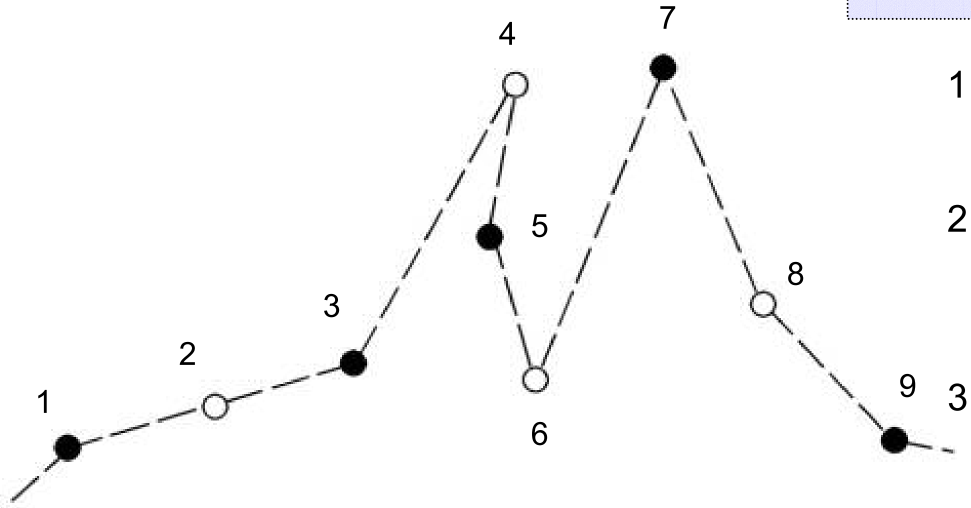
Roberto De Franceschi, DEI, University of Padua

Matteo Fischetti, DEI, University of Padua

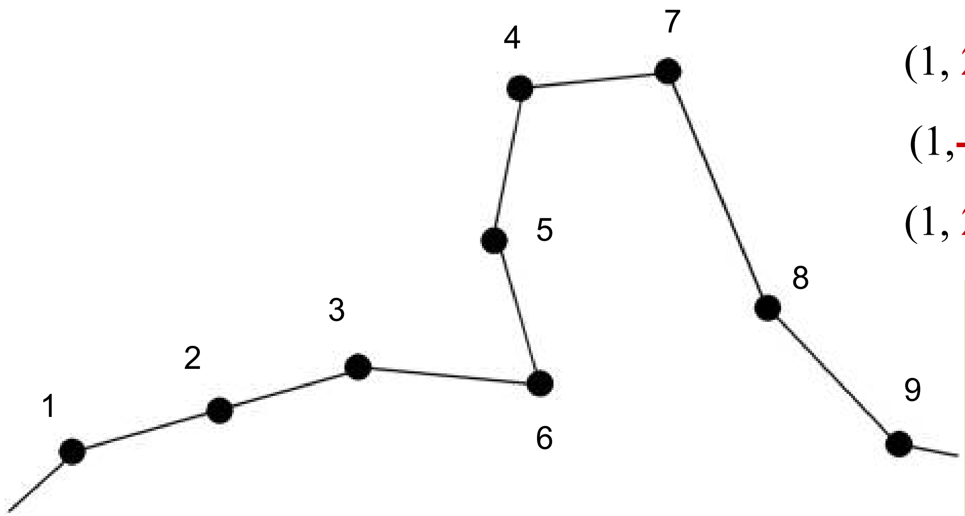
Paolo Toth, DEIS, University of Bologna

A method for the TSP (Sarvanov and Doroshko, 1981)

The ASSIGN *neighborhood*



1. consider a **given** tour as a sequence of nodes
2. fix the nodes in **odd** position, and remove the nodes in **even** position
3. Reassign the removed nodes in optimal way—an easy-solvable min-cost **assignment problem**



(1, **2**, 3, **4**, 5, **6**, 7, **8**, 9, …)

(1,**--**, 3, **--**,5, **--**,7, **--**, 9, …)

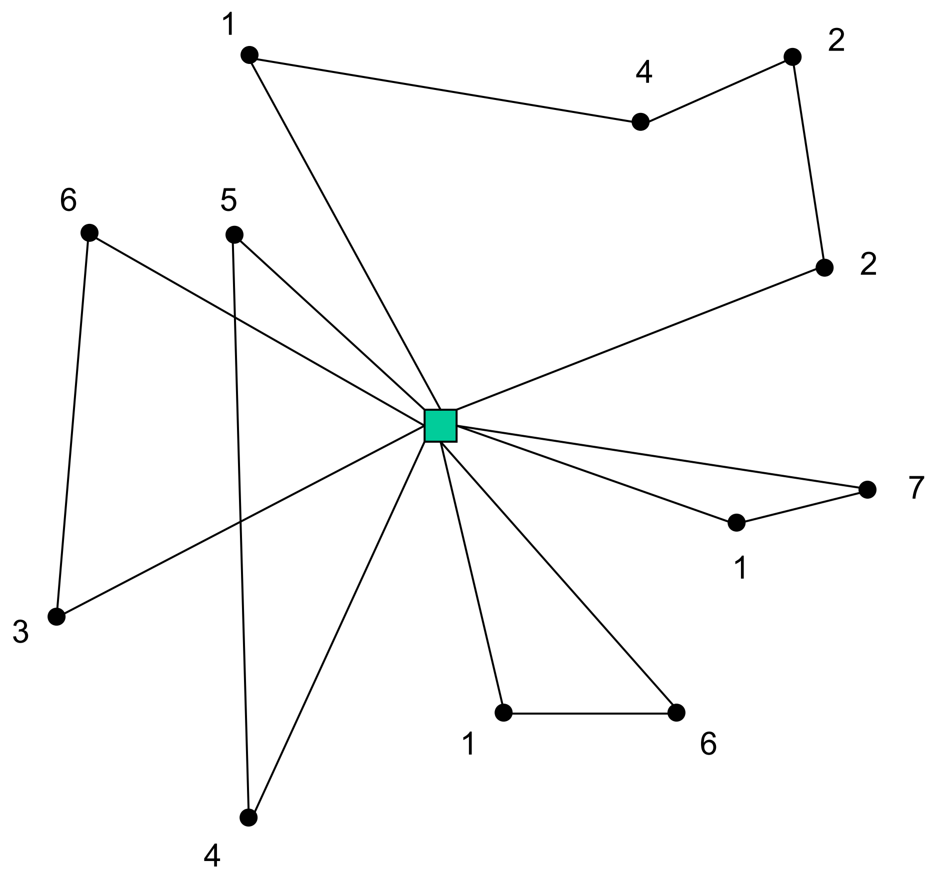
(1, **2**, 3, **6**, 5, **4**, 7, **8**, 9,…)

Neighborhood of **exponential cardinality** searchable in polynomial time, recently studied by:

Deineko and Woeginger (2000)

Firla, Spille and Weismantel (2002)

Capacitated Vehicle Routing Problem



Input

Depot

K vehicles

each with capacity *C*

N customers

with known demand d_i

Goal

K routes

not exceeding the given capacity

with minimum total cost

Capacitated Vehicle Routing Problem

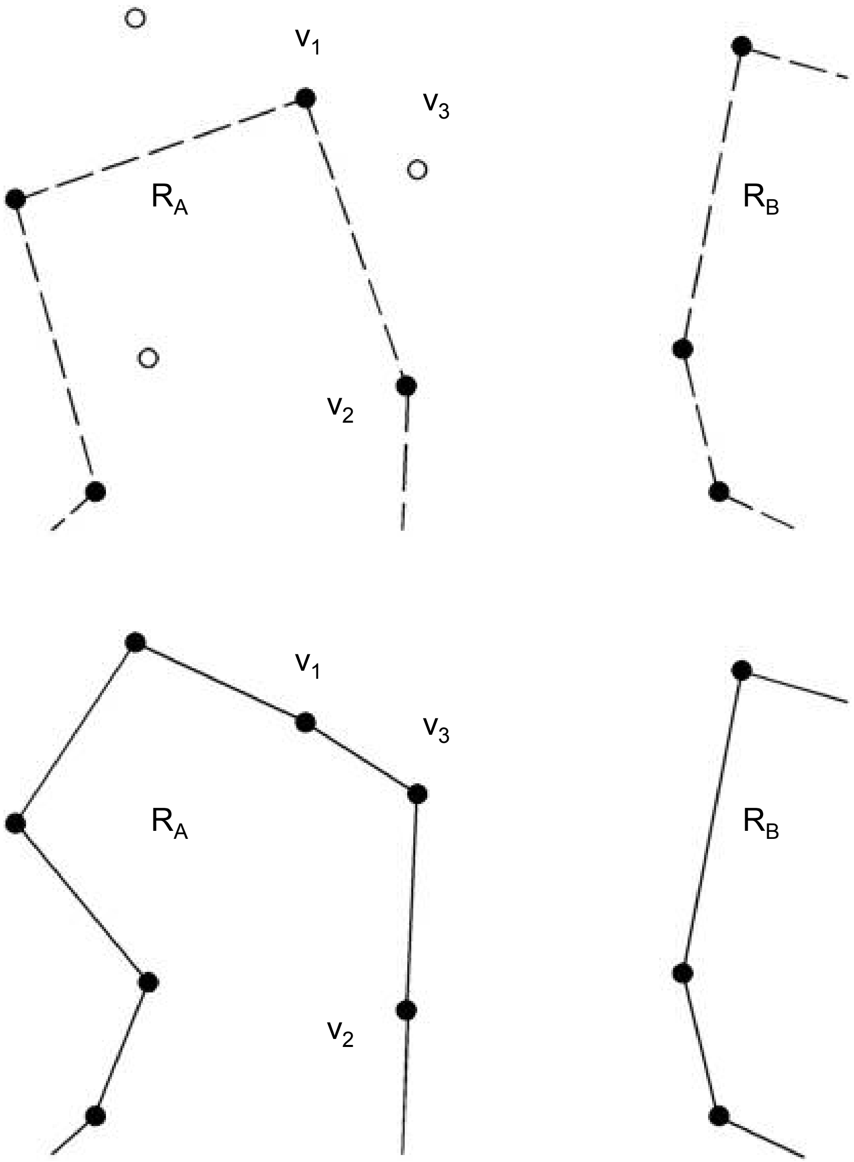
Selected literature on VRP heuristics

1959	Dantzig and Ramser: problem formulation
1964	Clarke and Wright: heuristic algorithm Balinski and Quandt: *set-partitioning* model
1976	Foster and Ryan: *Petal* heuristic
1981	Fisher and Jaikumar: *Generalized Assignment* heuristic
1993	Taillard: *Tabu Search* metaheuristic
1998	Toth and Vigo: *Granular Tabu Search* metaheuristic

Properties

- Important practical applications
- NP-hard
- Generalizes the *Traveling Salesman Problem (TSP)*

Basic extensions – Part I



Issue …

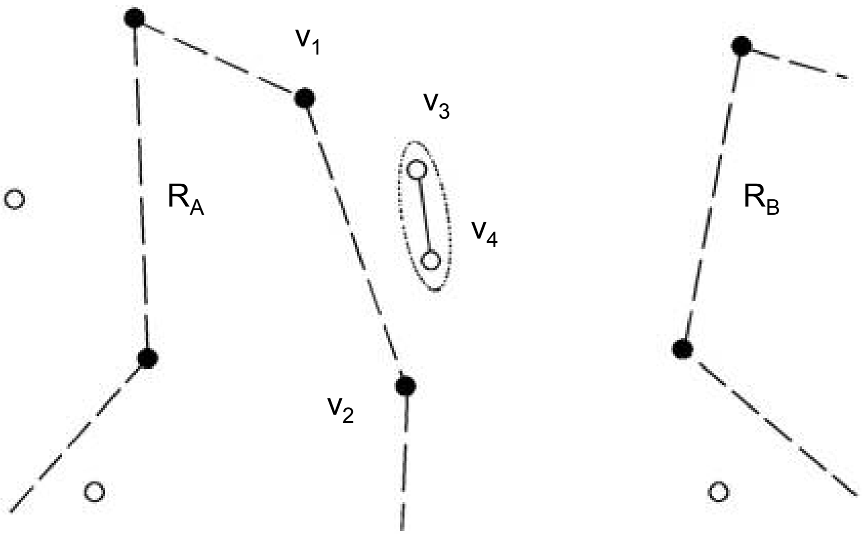
It seems useful to "move" node v_3 to route R_A (assuming this is feasible w.r.t.the capacity constraints)

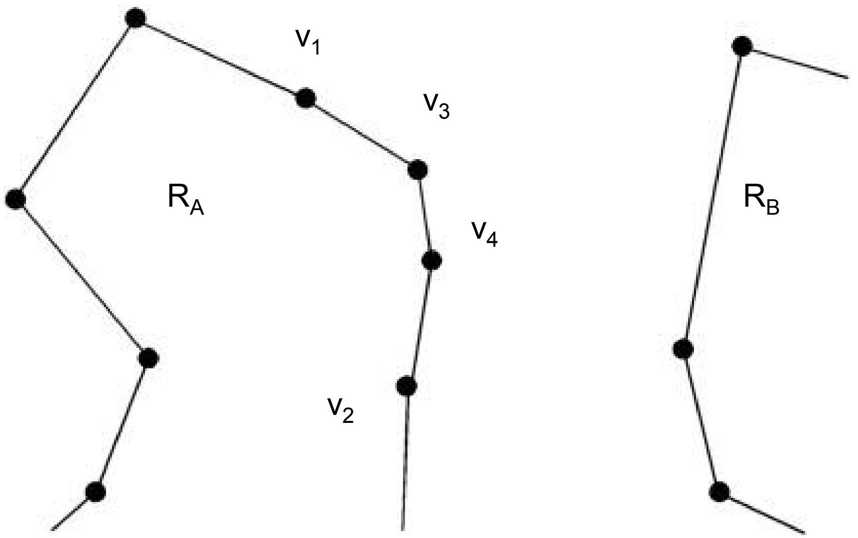
But … this cannot be done by a simple position-exchange between nodes

… solution

Introduce the concepts of ***restricted solution*** and ***insertion point***

Basic extensions – Part II





Issue …

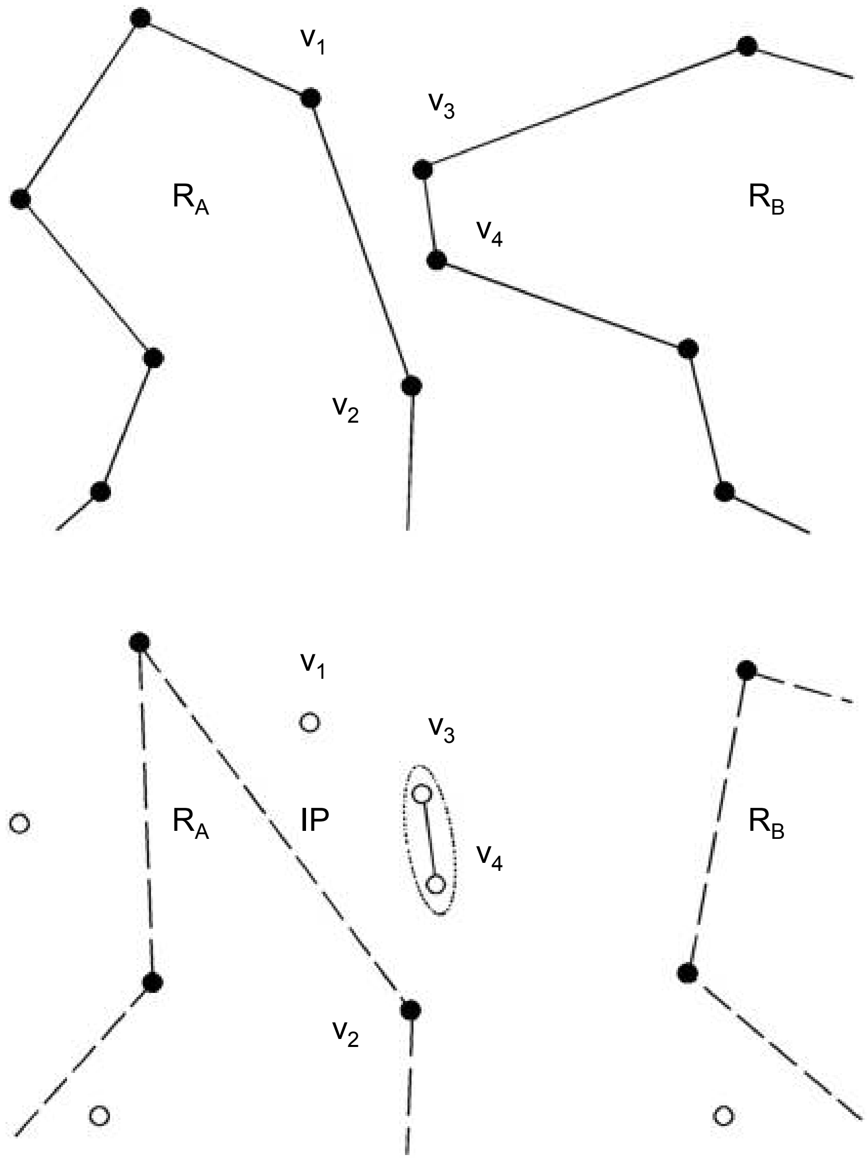
It seems useful to "move" **both** v_3 and v_4 to R_A (if feasible)

But … this cannot be done in one step by only "moving" single nodes

… solution

go beyond the basic odd/even scheme and introduce the notion of **extracted node sequences**

Basic extensions – Part III



Issue …

It is not possible to insert ***both*** v_1 and v_3-v_4 into the insertion point IP

… solution

generate a (possibly large) number of ***derived sequences*** through extracted nodes

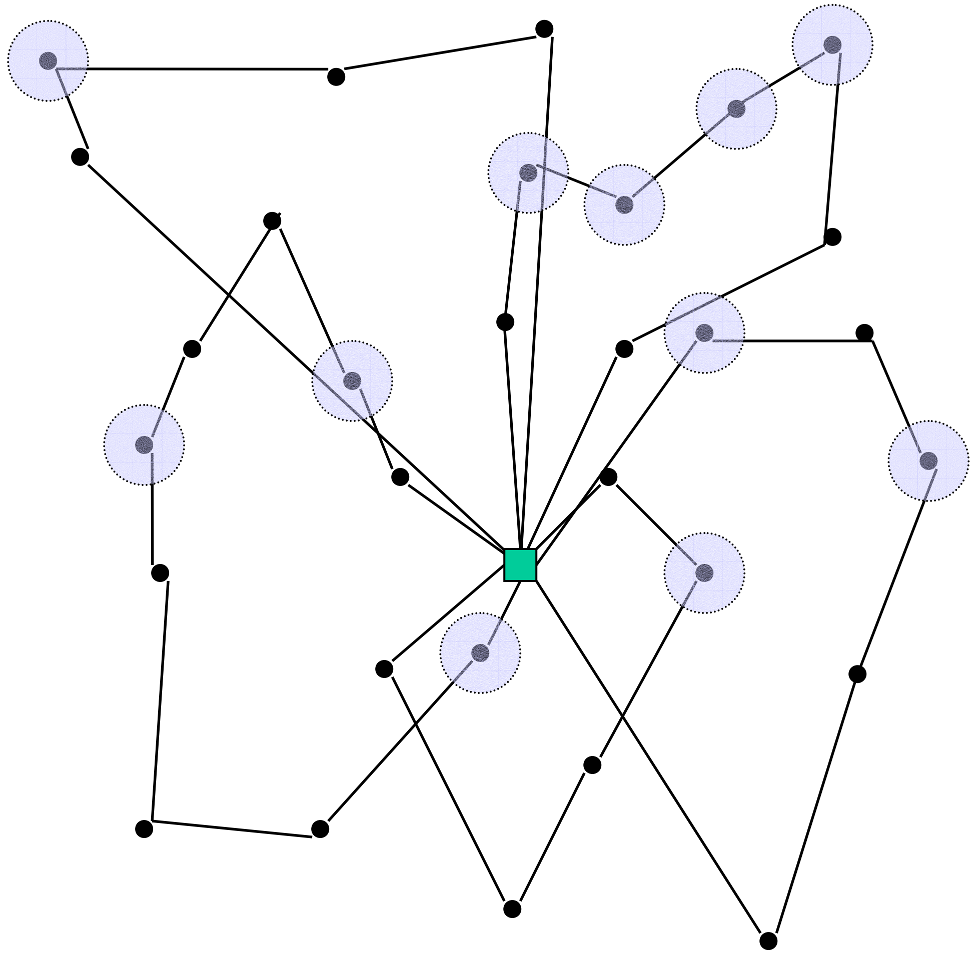
In the example, it is useful to generate the sequence v_1-v_3-v_4 to be placed in the insertion point IP

The SERR algorithm

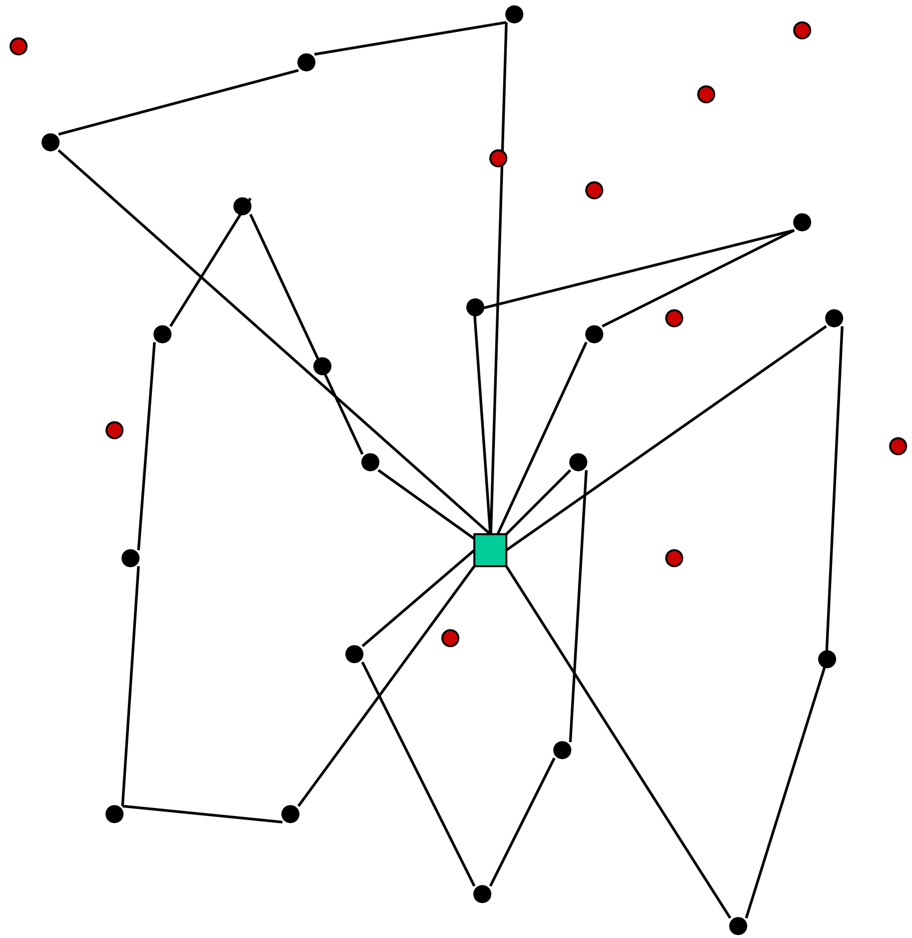
Steps

Initialization	generate, by any heuristic or metaheuristic, an initial solution
	Iteratively:
Selection	select the nodes to be extracted, according to suitable criteria (**schemes**)
Extraction	remove the selected nodes and generate the **restricted solution**
Recombination	starting from extracted nodes, generate a (possibly large) number of **derived sequences**
Re-insertion	re-insert a subset of the derived sequences into the restricted solution, in such a way that all the extracted nodes are covered again
Evaluation	verify a stopping condition and return, if it is the case, to the selection step

An example



An example



SERR Algorithm

Node re-insertion

Node re-insertion is done by solving the following ***set-partitioning*** model:

$$\min \sum_{s \in S} \sum_{i \in I} C_{si} x_{si}$$

$$\sum_{s \ni v} \sum_{i \in I} x_{si} = 1 \quad \forall v \text{ extracted}$$

$$\sum_{s \in S} x_{si} \leq 1 \quad \forall i \in I$$

$$d(r) + \sum_{s \in S} \sum_{i \in r} d(s) x_{si} \leq C \quad \forall r \in R$$

$$0 \leq x_{sj} \leq 1 \text{ integer} \quad \forall s \in S, \forall i \in I$$

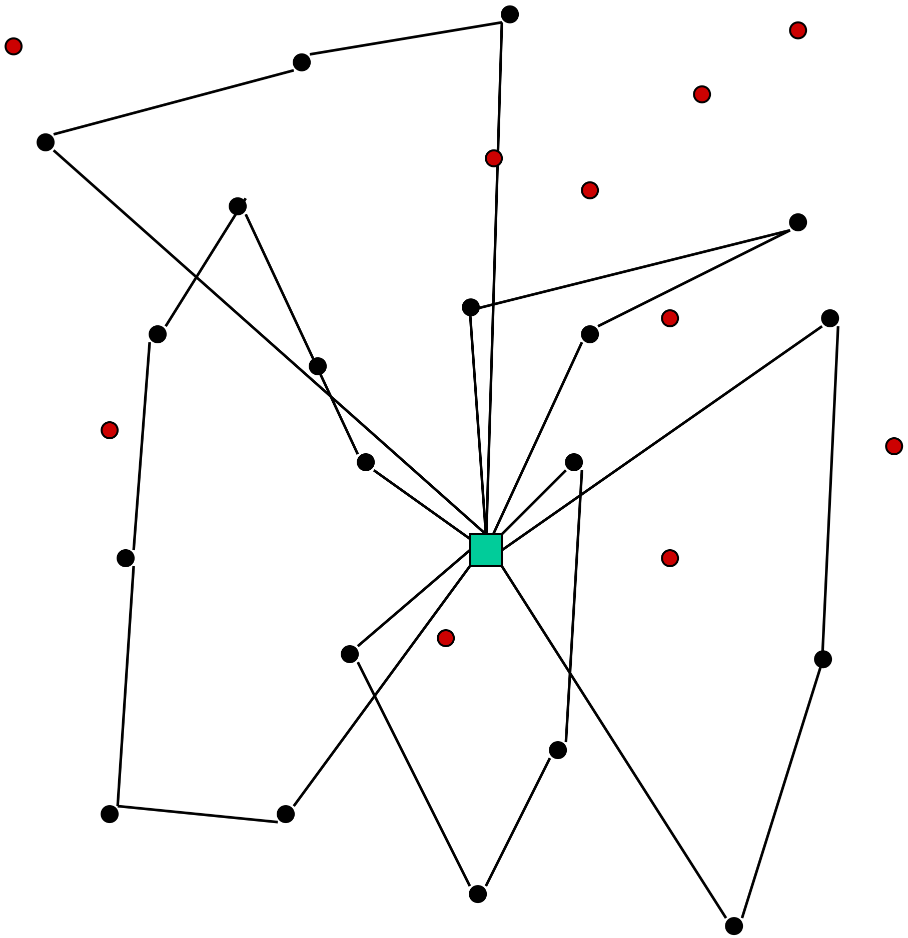
$x_{si} = 1$ if and only if sequence s goes into the insertion point i

C_{si} (best) insertion cost of sequence s into the insertion point i

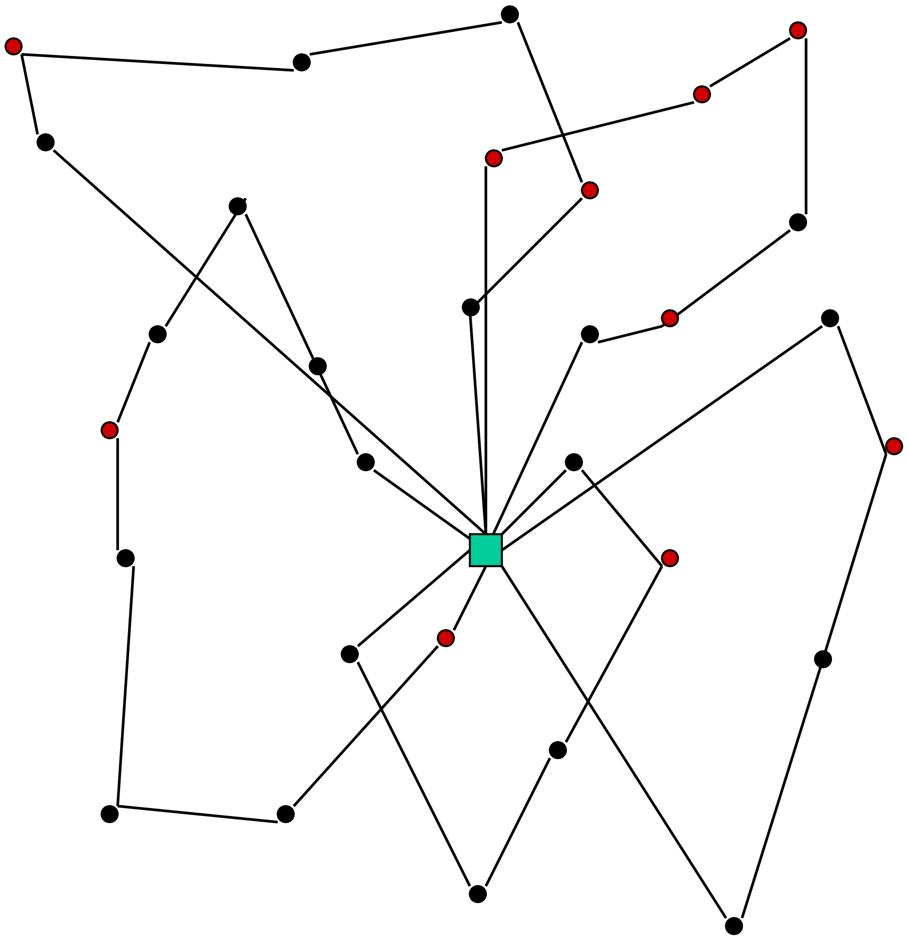
$d(r)$ total demand of the restricted route r

$d(s)$ total demand in the node sequence s

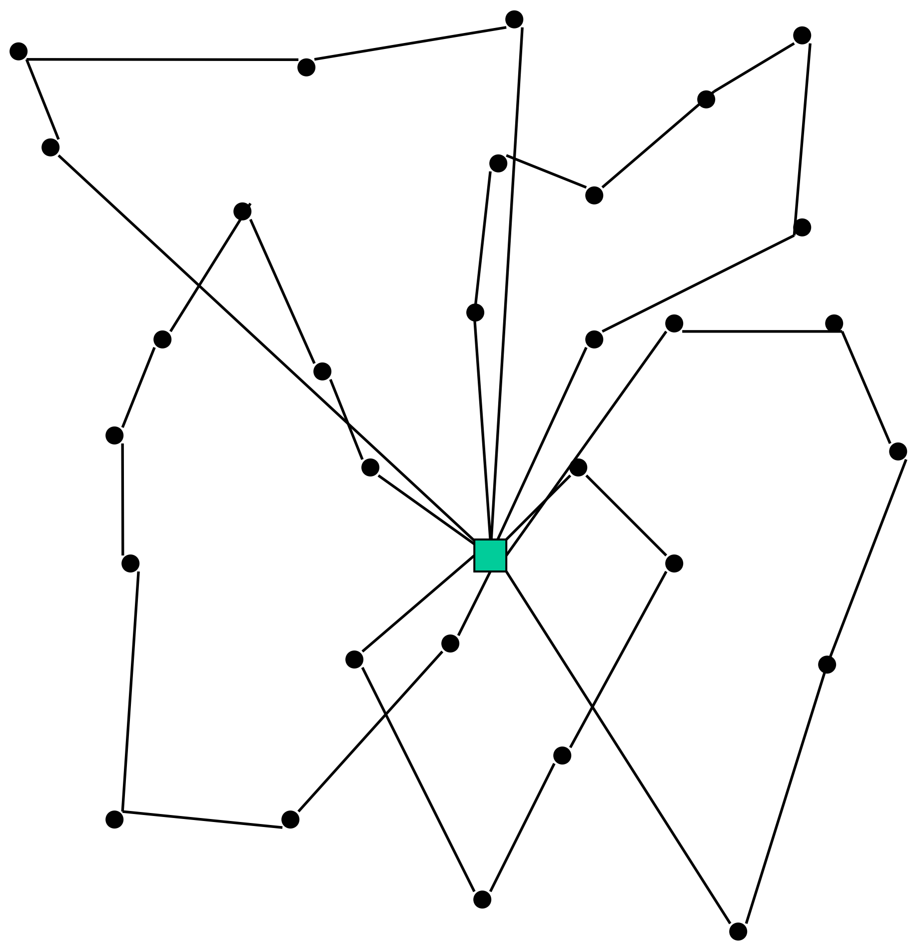
An example (cont.d)



An example (cont.d)

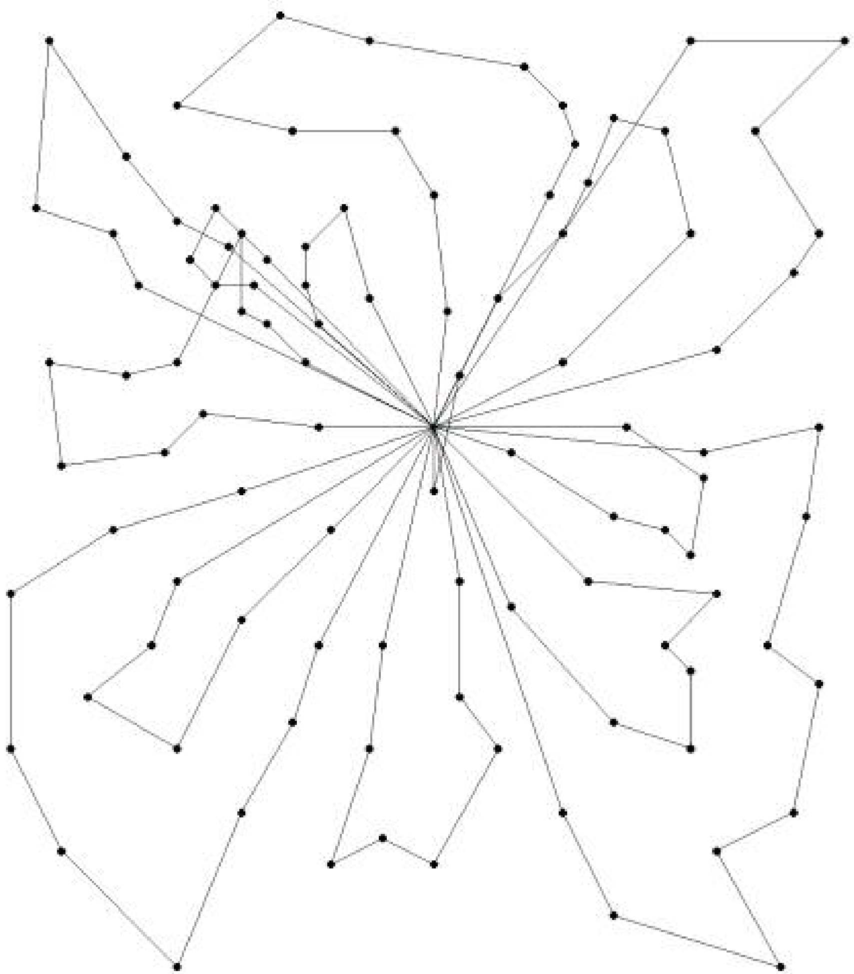


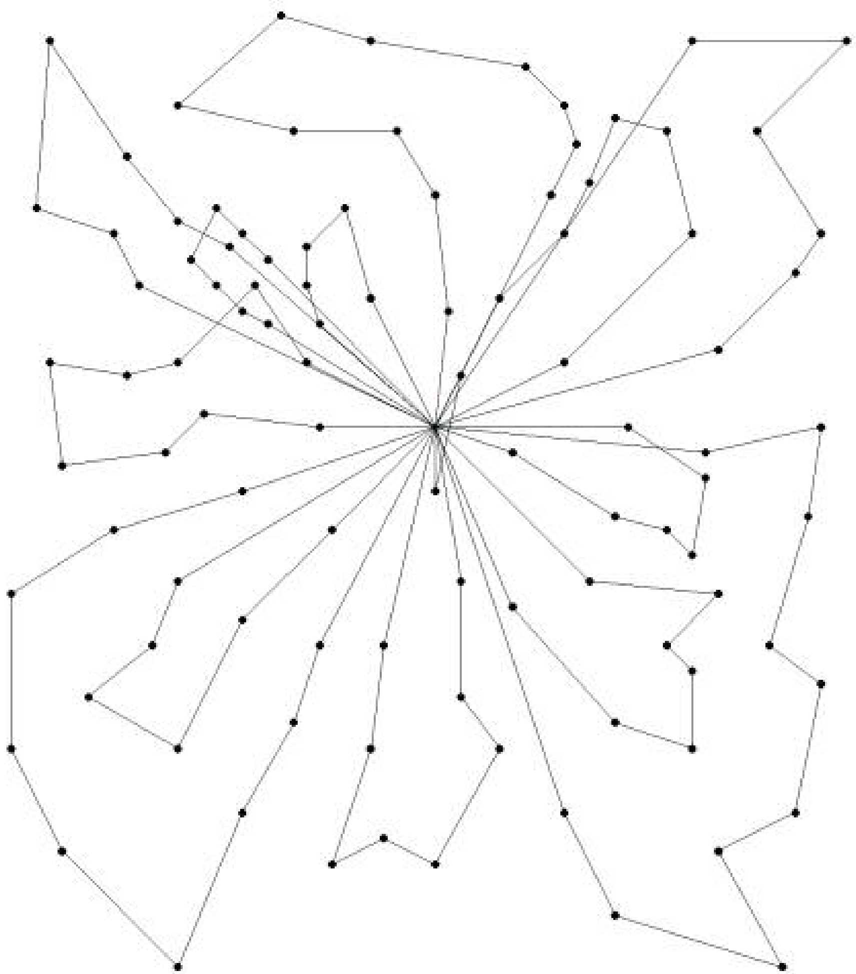
Initial Solution



Interesting solutions

Instance E-n101-k14 with rounded costs



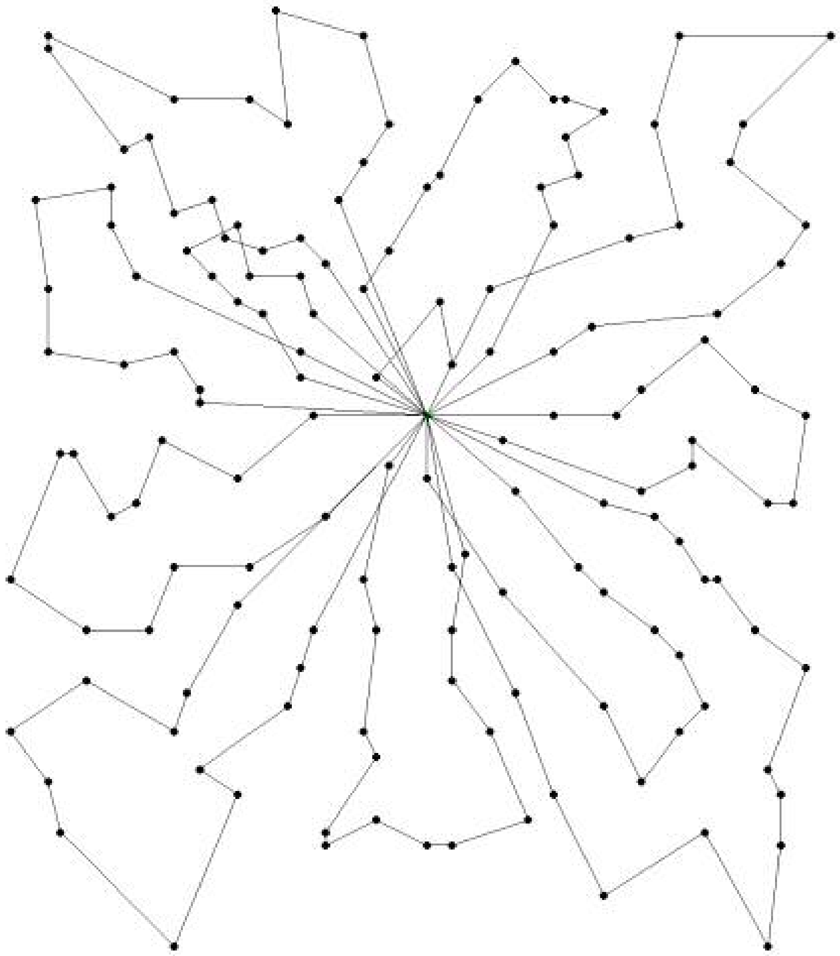


Initial solution: cost 1076
Xu and Kelly, 1996

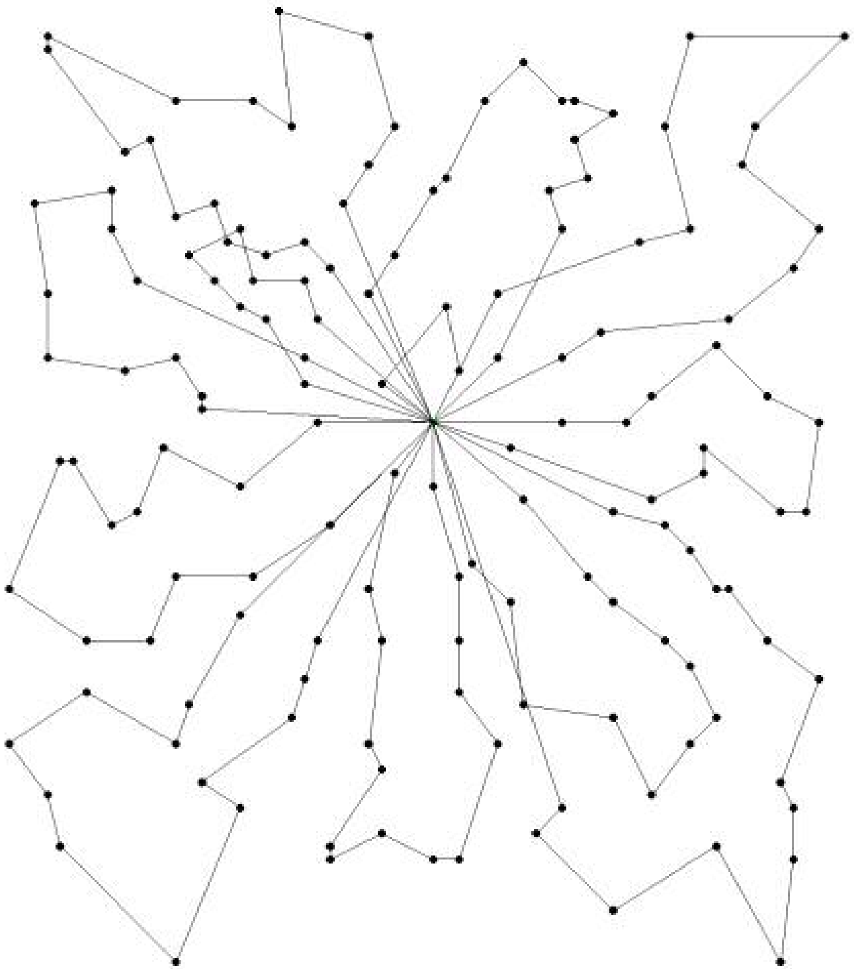
Final solution: cost 1067
New best known solution

Interesting solutions

Instance M-n151-k12 with rounded costs



Initial solution: cost 1023
Gendreau, Hertz and Laporte, 1996



Final solution: cost 1022
New best known solution

Some Computational Results

Instance	Optimal	*SERR* sol.	Gap	Time
P-n50-k8	631	631	0.00%	11:08
P-n55-k10	694	700	0.86%	16:50
P-n60-k10	744	744	0.00%	25:01
P-n60-k15	968	975	0.72%	12:27
P-n65-k10	792	796	0.51%	12:26
P-n70-k10	827	834	0.48%	50:08
B-n68-k9	1272	1275	0.24%	3:02:01
E-n51-k5	521	521	0.00%	4:30
E-n76-k7	682	682	0.00%	27:35
E-n76-k8	735	742	0.95%	30:39
E-n76-k10	830	835	0.60%	1:19:30
E-n76-k14	1021	1032	1.08%	2:45:20
E-n101-k8	815	820	0.61%	2:54:04
E051-05e	524.61	524.61	0.00%	4:51
E076-10e	835.26	835.32	< 0.01%	1:12:05
E101-08e	826.14	831.91	0.70%	2:30:55
E101-10c	819.56	819.56	0.00%	2:35:36
E-n101-k14	-	1076 -> 1067	-	1:36:05
M-n151-k12-a	-	1023 -> 1022	-	7:46:33

New best known solution

Optimal solution(*)

New best heuristic solution known

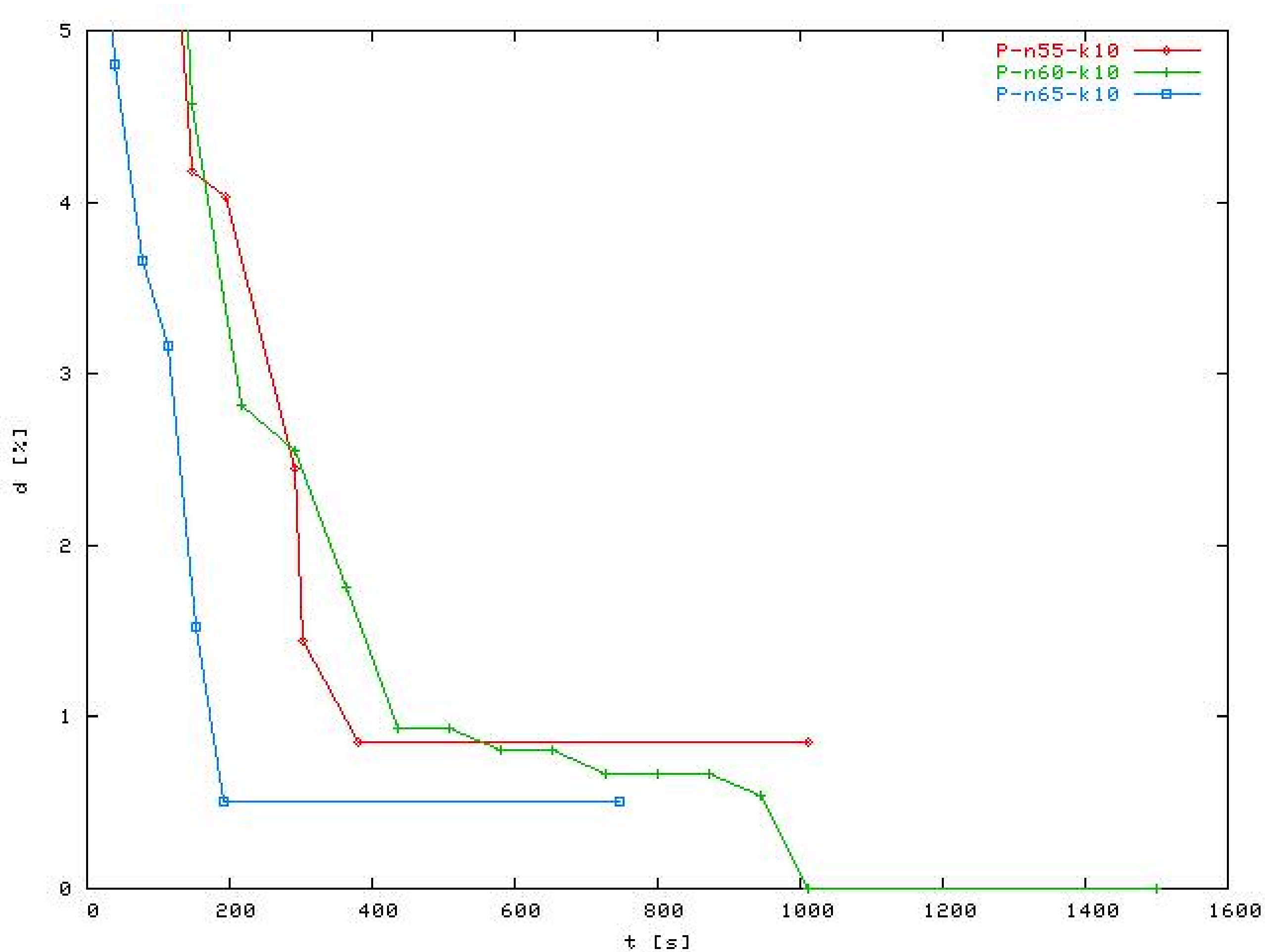
CPU times in the format [hh:]mm:ss

PC: Pentium M 1.6GHz

(*) Most optimal solutions have been found very recently by Fukasawa, Poggi de Aragao, Reis, and Uchoa (September 2003)

Results

Convergence properties of the SERR method



Low-cost solutions available in the first iterations

The best heuristics from the literature are credited for errors of about 2%

Conclusions

Achieved goals

1. **Definition** of a new **neighborhood** with exponential cardinality and of an effective (non-polynomial) **search algorithm**
2. **Simple implementation** based on a general ILP solver
3. **Evaluation** of the algorithm on a widely-used set of instances
4. Determination of the **new best solution** for two of the few instances not yet solved to optimality

Future directions of work

1. **Adaptation** of the method to more constrained versions of VRP, including VRP with **precedence constraints**
2. Use of an external **metaheuristic scheme**