A study on two query language approximations to the traveling salesman problem

version date: 5-4-2007

J.A. Bakker and J.H. ter Bekke [1]

Abstract

We discuss two query language approximations to the asymmetric traveling salesman problem (TSP). Both are based on the transformation of a directed cyclic geometric graph into a directed acyclic time graph for ride scheduling. This time graph represents the possible turns (arcs) between successive rides (nodes) on roads. The first approach applies the recursive *cascade* command of the Xplain database language and produces sometimes a correct shortest cycle, but in most cases it results in incorrect cycles passing some town more than once. The second approach does not apply the *cascade* command and produces correct cycles that are sometimes the shortest, but not in all cases. The estimated time complexity of both approaches is proportional to N^5, where N is the number of towns.

Keywords: graph reduction, query language, recursion, semantic data model, traveling salesman problem

1. Introduction

In previous work we described the application of the recursive *cascade* update command, the last language extension by the second author to the Xplain database language [9, 10]. This command can be applied to path problems in data structures corresponding with a weighed directed acyclic graph [1]. Examples using non-recursive data structures are: critical path analysis in project planning [14] and product planning [15]. We also demonstrated the applicability of this approach to problems related to recursive data structures such as the family tree [17, 18].

The algorithm underlying the *cascade* command attempts to achieve complete graph reduction as a preparation to a well ordered serial processing. However, complete graph reduction is impossible in the case of cyclic graphs [5]. Therefore the presence of any cycle is detected automatically by the Xplain-DBMS and the nodes involved in a cycle are reported to the user.

A first impression was that the *cascade* command could not be applied to real life networks such as road networks, because they are full of cycles. However, we demonstrated how a cyclic geometric graph for airports can be transformed into an acyclic graph for flight schedules [2], which is the basis for finding the fastest series of flights between two airports irrespective the number of transfers. A similar graph transformation applied to a road network produced an acyclic graph for ride scheduling: a database for roads and towns was extended with rides on roads and turns between successive rides. In this way shortest path problems in cyclic geometries became solvable [3].

Because of the polynomial time complexity [16] of the *cascade* operation, section 4 examines whether the application of the *cascade* command to TSP could lead to a correct result. In section 5 we also propose a greedy query language solution not using any *cascade* command.

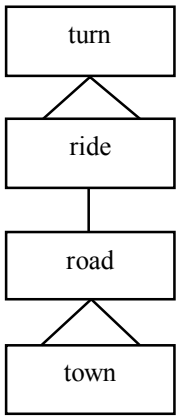
Section 2 presents a data model suitable for scheduling successive rides on roads, as presented earlier [3]. In section 3 we present an example of an acyclic scheduling graph derived from a cyclic geometry. Section 4 also describes how we generated the database for our research on TSP.

---

[1] This research started in the fall of 2003, was delayed by a re-organization within the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, the Netherlands, leading in July 2004 to the retirement of a number of staff members including the first author. The second author suddenly passed away on March 19th, 2004. Most of the publications related to semantic database principles and applications are available through: http://www.jhterbekke.net

2. A data model for scheduling rides and turns

A ride has geometric dimensions (occurs on a road) and a time dimension (time level) as well, whereas a turn denotes which ride may follow after a ride. The data model in figure 1, already applied in [3], enables us to register rides and turns between rides in a correct way.



type turn (i6) = previous_ride (i5), next_ride (i5).
type ride (i5) = road (i3), time_level (i2).
type road (i3) = from_town (a2), to_town (a2), distance (i3).
type town (a2) = name (a20).

Figure 1. A data model for rides and turns

In addition to the integrity rules inherently specified in the data model, more restrictions are required for data on turns. They are expressed by an assertion specifying the calculation of a derivable Boolean property 'turn *its* correctness' that must be true for all turns:

assert turn *its* correctness (*true*) = (next_ride *its* time_level = previous_ride *its* time_level + 1
and previous_ride *its* road <> next_ride *its* road
and previous_ride *its* road *its* to_town = next_ride *its* road *its* from_town
and previous_ride *its* road *its* from_town <> next_ride *its* road *its* to_town).

The last part of the assertion specifies that turnarounds (turns from a ride on road XY to a ride on road YX) must be absent in the database. In Xplain static restrictions, additional to the structural restrictions inherent in a data model, must be specified in terms of derived data, they cannot be specified in terms of the data model alone. This leads to a wealthy separation between inherent static constraints (referential and entity integrity rules) and additional static constraints and avoids the problems created by SQL. The last language is non-orthogonal since it allows for the specification of referential integrity rules by foreign key constraints and assertions as well. Consequently, in SQL it is the responsibility of the database manager to prevent the specification of overlapping or conflicting rules.

Turns could be derived from rides and rides from roads (by using join operations), but the Xplain language requires an explicit registration of rides and turns because this data language does not allow for join operations. The reason is that seemingly semantically correct join operations can lead to incorrect or incomplete query results as has been demonstrated for SQL, especially when set functions combined with the GROUP BY construct are involved [4, 11, and 13]. Therefore, only referential paths may be applied in Xplain and the direction of such paths

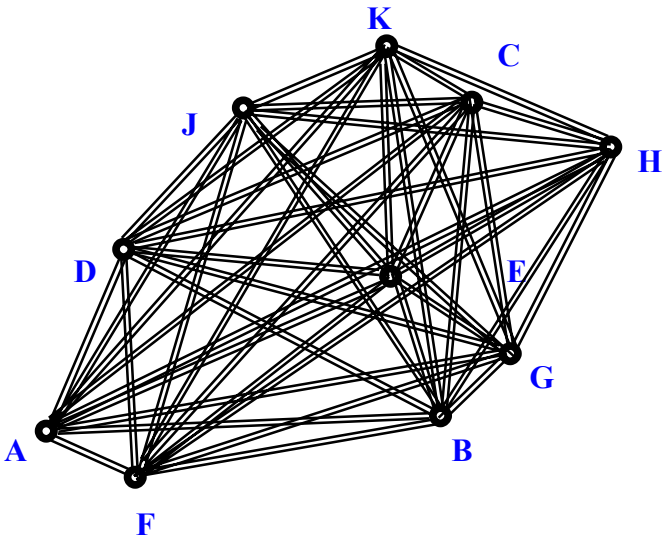
must be respected. An illustration is the following retrieval of the subset of connected towns where Amsterdam is the turning point between two rides:

get turn *its* previous_ride *its* road *its* from_town *its* name,
next_ride *its* road *its* to_town *its* name
where previous_ride *its* time_level = 1
and previous_ride *its* road *its* to_town *its* name = "Amsterdam".

The data definitions in figure 1 enable us presenting the required data on rides and turns as a directed acyclic graph: each turn (arc) always connects two successive rides (nodes) with successive time levels. The following section describes the application of the transformation of a cyclic road network into an acyclic scheduling graph, used for TSP-5.

3. Graph transformation

We illustrate our ideas by transforming the cyclic geometric graph of figure 2 into the acyclic time graph in figure 3 showing the possible turns between successive rides. This graph (a kind of Petri Net [6]) and the associated database can be used up to TSP-10.



Number of towns (N):	10
Number of time levels (N):	10
Number of one-direction roads (N*(N-1)):	90
Number of rides (N^2*(N-1)):	900
Number of rides with succeeding rides (N*$(N-1)^2$):	810
Number of usable turns ((N-2)*N*$(N-1)^2$):	6480

Figure 2. Example of a bi-directional road network between ten towns.

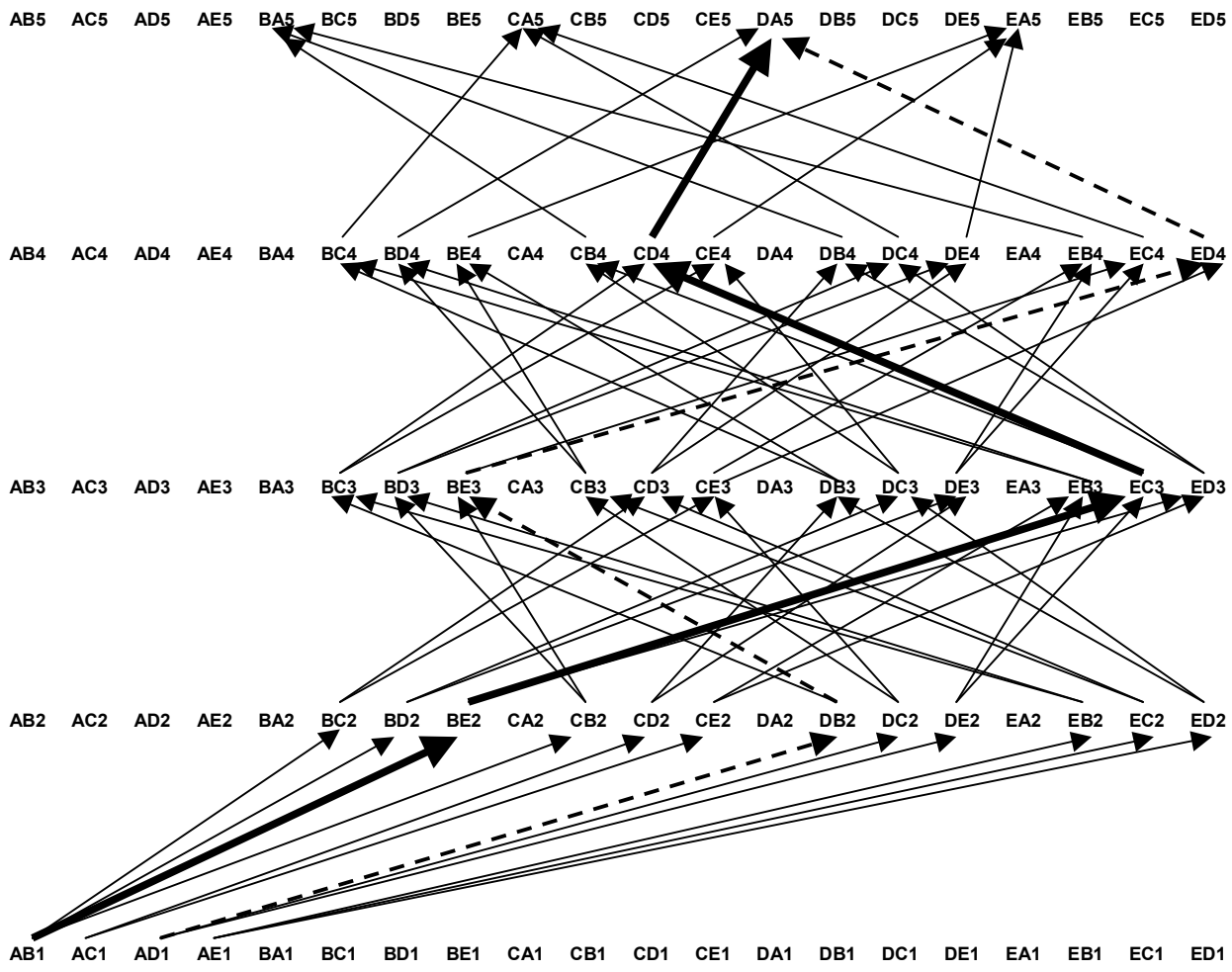


Figure 3. Rides and turns (arcs) starting in town A **(TSP-5,** geometry in figure 2).

Figure 3 shows the 24 possible round tours (4*3*2*1) through the towns A, B, C, D and E with town A as start and finish (TSP-5). For example, the geographic cycle ABCDEA is traversed by the following series of rides with successive time levels: AB1, BC2, CD3, DE4 and EA5. Later on it will become clear that the shortest cyclic tour can use the following sequence of rides: AB1, BE2, EC3, CD4 and DA5 (indicated by fat arrows in figure 3): ABECDA. The dotted arcs represent an example of an incorrect cycle formed by the rides AD1, DB2, BE3, ED4 and DA5: ADBEDA.

Table 1 shows the involved road data. The roads 1 - 20 are involved in finding a shortest cycle through the towns A, B, C, D and E. Table 2 shows data on the other roads. Both tables illustrate that we are dealing with the asymmetric TSP. *These both tables can be found in Appendix-I.*

Table 3 presents all 24 possible cycles for TSP-5 starting and finishing in town A. Of course each town can function as start and finish. We only have to shift the time levels in the desired direction in order to find equivalent cycles For example, if the route ABECDA is the shortest cycle, then BECDAB, CDABEC, DABECD and ECDABE are also shortest cycles.

time_ level = 1 ride	time_level = 2 ride	distance	time_level = 3 ride	distance	time_level = 4 ride	distance	time_level = 5 ride	distance	total distance
AB(49)	BC(39)	88	CD(48)	136	DE(34)	170	EA(46)	216	216
AB(49)	BC(39)	88	CE(24)	112	ED(33)	145	DA(23)	168	168
AB(49)	BD(44)	93	DC(47)	140	CE(24)	164	EA(46)	210	210
AB(49)	BD(44)	93	DE(34)	127	EC(24)	151	CA(66)	217	217
AB(49)	**BE(19)**	**68**	**EC(24)**	**92**	**CD(48)**	**140**	**DA(23)**	**163**	**163**
AB(49)	BE(19)	68	ED(33)	101	DC(47)	148	CA(66)	214	214
AC(67)	CB(40)	107	BD(44)	151	DE(34)	185	EA(46)	231	231
AC(67)	CB(40)	107	BE(19)	126	ED(33)	159	DA(23)	182	182
AC(67)	CD(48)	115	DE(34)	149	EB(20)	169	BA(50)	242	242
AC(67)	CD(48)	115	DB(43)	158	BE(19)	177	EA(46)	223	223
AC(67)	CE(24)	91	EB(20)	111	BD(44)	155	DA(23)	178	178
AC(67)	CE(24)	91	ED(33)	124	DB(43)	167	BA(50)	217	217
AD(25)	DB(43)	68	BC(39)	107	CE(24)	131	EA(46)	177	177
AD(25)	DB(43)	68	BE(19)	87	EC(24)	111	CA(66)	177	177
AD(25)	DC(47)	72	CB(40)	112	BE(19)	131	EA(46)	177	177
AD(25)	DC(47)	72	CE(24)	96	EB(20)	116	BA(50)	166	**166**
AD(25)	DE(34)	59	EB(20)	79	BC(39)	118	CA(66)	184	184
AD(25)	DE(34)	59	EC(24)	83	CB(40)	123	BA(50)	173	173
AE(47)	EB(20)	67	BC(39)	106	CD(48)	154	DA(23)	177	177
AE(47)	EB(20)	67	BD(44)	111	DC(47)	158	CA(66)	224	224
AE(47)	EC(24)	71	CB(40)	111	BD(44)	155	DA(23)	178	178
AE(47)	EC(24)	71	CD(48)	119	DB(43)	162	BA(50)	212	212
AE(47)	ED(33)	80	DB(43)	123	BC(39)	162	CA(66)	228	228
AE(47)	ED(33)	80	DC(47)	127	CB(40)	167	BA(50)	217	217

Table 3. All possible cycles for TSP-5 starting and finishing in town A

Apparently ABECDA is the shortest cycle and its total length is 163. The reverse cycle ADCEBA appears to have a total length 166.

4. Database generation

The Xplain language does not support the relational join operation, but the report generator of Xplain is able to execute join operations between the contents of reports produced by queries. Also the relational project and subtract operations can be applied to reports. In reports each line represents an instance of the set of query results; similar results (such as attribute values) are shown per column. Xplain also offers the possibility to apply sorting to reports on the bases of a selected column. Sorting is a well known preparative operation reducing the time complexity of joins. The result of a join is a new report. Any column in the resulting report can be removed (projection), which was applied to remove duplicate columns produced by a join operation. Queries producing rides and turns, using the facilities of the report generator, are shown in Appendix-I.

5. Applying the cascading update operation to TSP-5

In Appendix-II we present the results produced by a query using the recursive *cascade* command. In the case of attempting to find the shortest cycle through the towns A, B, C, D and E this query produces an incorrect 'shortest cycle' ADBEDA (total distance: 143). This result is incorrect

because town D is passed twice and town C is missed. This incorrect cycle is shown in figure 3 by dotted arcs. This incorrect result is quite understandable if we consider the essence of the applied cascading updates:

In the first part for each ride the distance 'ride *its* sdistance' to the starting town A is calculated step by step. First all rides get a distance inherited from the referenced road:

extend ride *with* distance = road *its* distance.

Then for all rides the value of 'ride *its* sdistance' is initialized using a value 'inf' that cannot be exceeded:

value inf = *total* road *its* distance.
extend ride *with* sdistance = inf.

A correction must be made for the rides with time level 1 and starting in town A:

update ride *its* sdistance = 0 *where* road *its* to_town = "A" *and* time_level = 1.

Using the cascading update operation the value of 'ride *its* sdistance' is calculated for successive rides (thus rides with higher time levels):

cascade ride *its* sdistance = *min* turn *its* previous_ride *its* sdistance + previous_ride *its* distance *where* c *per* next_ride.

Only correct turns may be applied. Therefore appendix-II also contains the specification of the derived attribute 'turn *its* c' defining the correctness of a turn in relationship to the problem at hand, but here we do not show how this derived attribute gets a value.

In a similar way we calculate for each ride the distance to the finish of the cycle (town A): 'ride *its* fdistance'.

Now we can calculate a total distance for each ride on a cyclic tour from town A to town A:

extend ride *with* totaldist = distance + sdistance + fdistance.

The minimal total distance can be determined now:

value minimum = *min* ride *its* totaldist.

Now we can retrieve the rides having this minimal total distance:

get ride *where* totaldist = minimum.

The relevant results of these two cascading update operation on the possibly rides with time levels 1 and 2 are summarized for TSP-5 in table 4 showing results that are incorrect because the correctness of the shortest pre-routes was ignored. An improved variant approach based on correct shortest pre-routes produced the results presented in table 5.

ride	ride *its* distance	ride *its* sdistance (shortest preceding route)	ride *its* (distance + sdistance)	ride *its* fdistance (shortest following route)	ride *its* (distance + fdistance)	ride *its* totaldist
AB1	49	0	49	114(BE2-EC3-CD4-DA5)	163	163
AC1	67	0	67	111(CE2-EB3-BD4-DA5)	178	178
AD1	25	0	25	118(DB2-BE3-ED4-DA5)#	143	**143**
AE1	47	0	47	129(EB2-BC3-CE4-EA5)# 129(EC2-CB3-BE4-EA5)#	176	176
BC2	39	49(AB1)	88	80(CE3-ED4-DA5)	119	168
BD2	44	49(AB1)	93	104(DE3-EB4-DA5) #	148	197
BE2	19	49(AB1)	68	95(EC3-CD4-DA5)	114	163
CB2	40	67(AC1)	107	75(BE3-ED4-DA5)	115	182
CD2	48	67(AC1)	115	104(DE3-EB4-BA5)	152	219
CE2	24	67(AC1)	91	87(EB3-BD4-DA5)	111	178
DB2	43	25(**AD1**)	68	75(BE3-ED4-DA5) #	118	**143**
DC2	47	25(AD1)	72	80(CE3-ED4-DA5) #	127	152
DE2	34	25(AD1)	59	87(EB3-BD4-DA5) #	121	146
EB2	20	47(AE1)	67	109(BC3-CE4-EA5) #	129	176
EC2	24	47(AE1)	71	105(CB3-BE4-EA5) #	129	176
ED2	33	47(AE1)	80	108(DB3-BE4-EA5) #	141	188
BC3	39	67(AE1-EB2)	106	70(CE4-EA5)	109	176
BD3	44	67(AE1-EB2)	111	80(DE4-EA5)	124	191
BE3	19	68(**AD1-DB2**)	87	56(ED4-DA5)	75	**143**
CB3	40	71(AE1-EC2)	111	65(BE4-EA5)	105	176
CD3	48	71(AE1-EC2)	119	80(DE4-EA5)	128	199
CE3	24	72(AD1-DC2)	96	56(ED4-DA5)	80	152
DB3	43	80(AE1-ED2)	123	65(BE4-EA5)	108	188
DC3	47	80(AE1-ED2)	127	70(CE4-EA5)	117	197
DE3	34	93(AB1-BD2)	127	70(EB4-BA5)	104	197
EB3	20	59(AD1-DE2)	79	67(BD4-DA5)	87	146
EC3	24	59(AD1-DE2)	83	71(CD4-DA5)	95	154
ED3	33	68(AB1-BE2)	101	93(DB4-BA5)	126	194
BC4	39	79(AD1-DE2-EB3)	118	66(CA5)	105	184
BD4	44	79(AD1-DE2-EB3) #	123	23(DA5)	67	146
BE4	19	111(AE1-EC2-CB3) #	130	46(EA5)	65	176
CB4	40	83(AD1-DE2-EC3)	123	50(BA5)	90	173
CD4	48	83(AD1-DE2-EC3) #	131	23(DA5)	71	154
CE4	24	106(AE1-EB2-BC3) #	130	46(EA5)	70	176
DB4	43	101(AB1-BE2-ED3) #	144	50(BA5)	93	194
DC4	47	101(AB1-BE2-ED3)	148	66(CA5)	113	214
DE4	34	119(AE1-EC2-CD3) #	153	46(EA5)	80	199
EB4	20	96(AD1-DC2-CE3)	116	50(BA5)	70	166
EC4	24	87(AD1-DB2-BE3)	111	66(CA5)	90	177
ED4	33	87(**AD1-DB2-BE3**) #	120	23(DA5)	56	**143**
BA5	50	116(AD1-DC2-CE3-EB4)	166	0	50	166
CA5	66	111(AD1-DB2-BE3-EC4)	177	0	66	177
DA5	23	120(**AD1-DB2-BE3-ED4**)	143	0	23	**143**
EA5	46	130(AE1-EC2-CB3-BE4)# 130(AE1-EB2-BC3-CE4)#	176	0	46	176

Table 4. Results of the cascading update for TSP-5 using some incorrect pre-routes (#)

ride	ride *its* distance	ride *its* sdistance (shortest correct preceding route)	ride *its* (distance + sdistance)	ride *its* fdistance (shortest correct following route)	ride *its* (distance + fdistance)	ride *its* totaldist
AB1	**49**	**0**	**49**	**114(BE2-EC3-CD4-DA5)**	**163**	**163**
AC1	67	0	67	111(CE2-EB3-BD4-DA5)	178	178
AD1	25	0	25	152(DB2-BC3-CE4-EA5)	177	177
AE1	47	0	47	inf(none)	inf	inf
BC2	39	49(AB1)	88	80(CE3-ED4-DA5)	119	168
BD2	44	49(AB1)	93	104(DE3-EB4-BA5)	148	197
BE2	19	49(AB1)	68	95(EC3-CD4-DA5)	114	163
CB2	40	67(AC1)	107	75(BE3-ED4-DA5)	115	182
CD2	48	67(AC1)	115	104(DE3-EB4-BA5)	152	219
CE2	24	67(AC1)	91	87(EB3-BD4-DA5)	111	178
DB2	43	25(AD1)	68	109(BC3-CE4-EA5)	152	177
DC2	47	25(AD1)	72	105(CB3-BE4-EA5)	152	177
DE2	34	25(AD1)	59	inf(none)	inf	inf
EB2	20	47(AE1)	67	inf(none)	inf	inf
EC2	24	47(AE1)	71	inf(none)	inf	inf
ED2	33	47(AE1)	80	inf(none)	inf	inf
BC3	39	67(AE1-EB2)	106	70(CE4-EA5)	109	176
BD3	44	67(AE1-EB2)	111	80(DE4-EA5)	124	191
BE3	19	68(AD1-DB2)	87	56(ED4-DA5)	75	143
CB3	40	71(AE1-EC2)	111	65(BE4-EA5)	105	176
CD3	48	71(AE1-EC2)	119	80(DE4-EA5)	128	199
CE3	24	72(AD1-DC2)	96	56(ED4-DA5)	80	152
DB3	43	80(AE1-ED2)	123	65(BE4-EA5)	108	188
DC3	47	80(AE1-ED2)	127	70(CE4-EA5)	117	197
DE3	34	93(AB1-BD2)	127	70(EB4-BA5)	104	197
EB3	20	59(AD1-DE2)	79	67(BD4-DA5)	87	146
EC3	24	59(AD1-DE2)	83	71(CD4-DA5)	95	154
ED3	33	68(AB1-BE2)	101	93(DB4-BA5)	126	194
BC4	39	79(AD1-DE2-EB3)	118	66(CA5)	105	184
BD4	44	111(AE1-EC2-CB3)	155	23(DA5)	67	178
BE4	19	inf(none)	inf	46(EA5)	65	inf
CB4	40	83(AD1-DE2-EC3)	123	50(BA5)	90	173
CD4	48	106(AE1-EB2-BC3)	154	23(DA5)	71	177
CE4	24	inf(none)	inf	46(EA5)	70	inf
DB4	43	119(AE1-EC2-CD3)	162	50(BA5)	93	212
DC4	47	101(AB1-BE2-ED3)	148	66(CA5)	113	214
DE4	34	inf(none)	inf	46(EA5)	80	inf
EB4	20	96(AD1-DC2-CE3)	116	50(BA5)	70	166
EC4	24	87(AD1-DB2-BE3)	111	66(CA5)	90	177
ED4	33	inf(none)	inf	23(DA5)	56	inf
BA5	**50**	**116(AD1-DC2-CE3-EB4)**	**166**	**0**	**50**	**166**
CA5	66	111(AD1-DB2-BE3-EC4)	177	0	66	177
DA5	23	154(AE1-EB2-BC3-CD4)	177	0	23	177
EA5	46	inf (none)	inf	0	46	inf

Table 5. Results of the cascading update for TSP-5 using correct pre-routes

In table 4 we see that many routes preceding a ride at time level 4 are marked with "#". This indicates that the involved preceding route is incorrect because some town is visited more than once. Moreover, many rides at time level 4 do not have any correct preceding route at all.

Table 5 shows many rides without any correct preceding route or without any correct succeeding route, which is indicated by 'inf'. An interesting observation in table 5 is that ride AB1 is followed by the shortest route BE2-EC3-CD4-DA5. Apparently the shortest cycle ABECDA (163) can be found if the query is improved by using only shortest succeeding rides.

In a similar way ride BA5 is preceded by the shortest route AD1-DC2-CE3-EB4. In this way the reverse cycle ADCEBA (166) of the shortest cycle can be found.

Because of these results, we also studied a greedy approach not using the *cascade* command [section 6].

6. A greedy approach

The number of repetitive actions in TSP is predictable for each chosen number of towns; therefore there is no need to apply any recursive command. The greedy approach starts with operations on rides with time level 2. Using data on turns, the query determines for each of these rides the shortest preceding ride with time level 1 starting in town A and copies the starting town (here A) of the relevant ride 1 to the corresponding ride 2. The distance traversed by the shortest preceding ride 1 (starting in A) is added to the distance traversed by ride 2 using a derived attribute 'ride *its* sdistance' (distance to start town). Similar steps are repeated for the rides with time levels 3-5. This query and its results for diverse inputs are shown in Appendix-III.

Here we discuss these results in more detail for finding the shortest cycle starting and finishing in town A and passing through the towns B, C, D and E:

The **first** step produced the following pre-routes over three towns (incl. its sdistance):

AB1(49)	+ BC2(39) =	ABC(88)
	+ BD2(44) =	ABD(93)
	+ BE2(19) =	**ABE**(68)
AC1(67)	+ CB2(40) =	ACB(107)
	+ CD2(48) =	ACD(115)
	+ CE2(24) =	ACE(91)
AD1(25)	+ DB2(43) =	ADB(68)
	+ DC2(47) =	ADC(72)
	+ DE2(34) =	ADE(59)
AE1(47)	+ EB2(20) =	AEB(67)
	+ EC2(24) =	AEC(71)
	+ ED2(33) =	AED(80)

The query produced twelve rides at time level 2 with a preceding ride starting in A. This equals the number of possible tours over three towns starting in A: (5-1)*(5-2) = 12. This list of tours over three towns is correct and complete, so at this stage there is no loss of information.

The **second** step produced the following pre-routes over four towns (incl. its sdistance):

ABC(88)	is not the shortest pre-route for ride CD3 or CE3.		
ABD(93)	+ DE3(34)	=	ABDE(127)
ABE(68)	+ ED3(33)	=	ABED(101)
	is not shortest pre-route for ride EC3.		
ACB(107)	is not shortest pre-route for ride BD3 or ride BE3.		
ACD(115)	is not shortest pre-route for ride DB3 or ride DE3.		
ACE(91)	is not shortest pre-route for ride EB3 or ride ED3.		
ADB(68)	+ BE3(19)	=	ADBE(87)
ADC(72)	+ CE3(24)	=	ADCE(96)
ADE(59)	+ EB3(20)	=	ADEB(79)
	+ **EC**3(24)	=	ADEC(83)
AEB(67)	+ BC3(39)	=	AEBC(106)
	+ BD3(44)	=	AEBD(111)
AEC(71)	+ CB3(40)	=	AECB(111)
	+ CD3(48)	=	AECD(119)
AED(80)	+ DB3(43)	=	AEDB(123)
	+ DC3(47)	=	AEDC(127)

In the greedy approach, the route ABE is not the shortest correct pre-route for ride EC3 (but it is tour ADE), so we are not able to find the shortest cycle ABECDA in this way. Now the query produced twelve results, which is half the number of possible routes over four towns starting in the same town: (5-1)(5-2)(5-3) = 24. Because of this loss of information there is no guarantee that the shortest cycle is found. The average probability that the shortest pre-route is found in this algorithm is 50%.

After the **third** step the following pre-routes over five towns (sdistance) are produced:

ABDE(127)	is not the shortest pre-route for ride EC4.		
ABED(101)	+ DC4(47)	=	ABEDC(148)
ADBE(87)	+ EC4(24)	=	ADBEC(111)
ADCE(96)	+ EB4(20)	=	ADCEB(116)
ADEB(79)	+ BC4(39)	=	ADEBC(118)
ADEC(83)	+ CB4(40)	=	ADECB(123)
AEBC(106)	+ CD4(48)	=	AEBCD(154)
AEBD(111)	is not the shortest pre-route for ride DC4.		
AECB(111)	+ BD4(44)	=	AECBD(155)
AECD(119)	+ DB4(43)	=	AECDB(162)
AEDB(123)	is not the shortest pre-route for ride BC4.		
AEDC(127)	is not the shortest pre-route for ride CB4.		

The query produced eight routes over five towns starting in A, which means that 8/24 = 33.3 % of the twenty four possible routes are found.

After the **fourth** step the following cycles are produced:

ABEDC(148)	is not the shortest pre-route for ride CA5.			
ADBEC(111)	+ CA5(66)	=	ADBECA(177)	
ADCEB(116)	**+ BA5(50)**	**=**	**ADCEBA(166)**	(**not the shortest cycle, but its reverse**).
ADEBC(118)	is not the shortest pre-route for ride CA5.			
ADECB(123)	is not the shortest pre-route for ride BA5.			
AEBCD(154)	+ DA5(23)	=	AEBCDA(177)	
AECBD(155)	is not the shortest pre-route for ride DA5.			
AECDB(162)	is not the shortest pre-route for ride BA5.			

These final results are complying with table 5: ride BA5 has the shortest correct preceding route constituted by the rides AD1, DC2, CE3 and EB4. Only 3 of the possible 24 cycles are produced; the chance to find the shortest cycle is 1/8 = 12.5 %. Other examples for TSP-5 in Appendix-II show that sometimes four cycles are produced; a success probability of at most 1/6 = 16.7 %. However, in the case of D as start/finish the same query produced the shortest cyclic route DABECD (163), which is equivalent to the route ABECDA (163). Apparently, the choice of the start/finish town can influence the results. Using similar queries for finding cycles over more than five towns, the probability of finding a correct cycle by the greedy approach decreased with the number of involved towns.

7. Discussion

Since decennia TSP is a famous ***NP***-complete problem [7]. Considered as a combinatorial optimization problem its complexity is proportional to ***(N-1)!***, where ***N*** is the number of towns to be visited. Our work on TSP-5, using the *cascade* command, demonstrates that query language specifications in combination with an acyclic scheduling graph sometimes lead to a polynomial solution because an exhaustive construction and comparison of all alternative routes is avoided. A disadvantage of applying the cascading update operation to TSP is that it in most cases produces incorrect cycles.

Therefore we also examined another greedy approach not using the *cascade* command. The last approach produces correct cycles, sometimes even the shortest cycle, but it does not guarantee that the shortest cycle is found. The probability of finding the correct solution will decrease with the number of involved towns.

Finally, as shown in Appendix-IV, both discussed query language approaches to TSP have an estimated time complexity of $O\{N^5\}$. The costs of deriving data on rides and turns from the geometric data on roads are proportional to N^4, but this is a single activity to create our database.

References

[1] J.M. Aldous and R.J. Wilson, Graphs and Applications, an Introductory Approach, Springer-Verlag, London - Berlin - Heidelberg (2000).

[2] J.A. Bakker and J.H. ter Bekke, A Query Language Solution For Fastest Flight Connections, Proceedings International Conf. on Databases and Applications DBA 2004, Innsbruck, Austria, M.H. Hamza (Ed.), ACTA Press, Anaheim - Calgary - Zürich (2004), pp. 197-202.

[3] J.A. Bakker and J.H. ter Bekke, A Query Language Solution For Shortest Path Problems In Cyclic Geometries, Proceedings International Conf. on Databases and Applications DBA 2004, Innsbruck, Austria, M.H. Hamza (Ed.), ACTA Press, Anaheim-Calgary -Zürich (2004), pp. 203-207.

[4] Bert Bakker and Johan ter Bekke, Fool Proof Query Access to Search Engines, Proc. 3rd Int. Conf. on Information Integration and Web-based Applications & Services, Linz, Austria, W.Winiwarter, S. Bressan and I.K. Ibrahim (Eds.), Österreichisches Computer Gesellschaft (2001), p.389-394.

[5] J. Bang-Jensen and G. Gutin, Digraphs: Theory, Algorithms and Applications, Springer-Verlag, London (2001).

[6] A.A.S. Danthine, Protocol Representation with Finite-State Models, IEEE Transactions on Communication, Vol. COM-20 (1980), pp. 632-643.

[7] G. Gutin and A.P. Punnen (eds.), Combinatorial Optimization, Vol. 12: The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, Boston, London (2002).

[8] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (Eds.), The Traveling Salesman Problem, a Guided Tour of Combinatorial Optimization, John Wiley & Sons, Chicester - New York - Brisbane - Toronto - Singapore (1985).

[9] F. Rolland, The essence of databases, Prentice Hall, Hemel Hempstead (1998).
[10] J.H. ter Bekke, Semantic Data Modeling, Prentice Hall, Hemel Hempstead (1992).
[11] J.H. ter Bekke, Can we rely on SQL?, Proceedings 8th International DEXA Workshop, Toulouse (1997), R.R. Wagner (Ed.), IEEE Computer Society, pp. 378-383.
[12] J.H. ter Bekke, Advantages of a compact semantic meta model, Proceedings 2nd IEEE Metadata Conference, Silver Spring, USA (1997), http://www.computer.org/conferen/proceed/meta97/papers/jterbekke/jterbekke.html
[13] J.H. ter Bekke and J.A. Bakker, Limitations of relationships constructed from coinciding data, Proceedings International Conference on Intelligent Systems and Control (ICSC 2001), Clearwater, Florida, USA, H.M. Hamza (Ed.), ACTA Press, Anaheim - Calgary – Zürich (2001, pp. 247-252.
[14] J.H. ter Bekke and J.A. Bakker, Content-driven specifications for recursive project planning applications, Proceedings International Conference on Applied Informatics (AI 2002), Innsbruck, Austria, M.H. Hamza (Ed.), ACTA Press, Anaheim - Calgary - Zürich (2002), pp. 448-452.
[15] J.H. ter Bekke and J.A. Bakker, Recursive queries in product databases, Flexible Query Answering Systems, Proc. 5th Int. Conf. on Flexible Query Answering Systems 2002, Copenhagen, Denmark, October 27-29, 2002, LNCS (Subseries LNAI), Vol. 2522, T. Andreasen, A. Motro, H. Christiansen and H. Legind Larsen (Eds.), Springer-Verlag, Berlin - Heidelberg (2002), pp. 44-55.
[16] J.H. ter Bekke and J.A. Bakker, Fast Recursive Data Processing in Graphs Using Reduction, Proceedings International Conference on Applied Informatics (AI 2003), Innsbruck, Austria, M.H. Hamza (Ed.), ACTA Press, Anaheim - Calgary - Zürich (2003), pp. 490-494.
[17] J.H. ter Bekke and J.A. Bakker, Modeling and Querying Recursive Data Structures I: Introduction, Proceedings Int. Conf. on Artificial Intelligence and Soft Computing ASC 2004, Banff, Canada, H. Leung (Ed.), ACTA Press, Anaheim - Calgary - Zürich (2003), pp. 278-282.
[18] J.H. ter Bekke and J.A. Bakker, Modeling and Querying Recursive Data Structures II: A Semantic Approach, Proc. Int. Conf. on Artificial Intelligence and Soft Computing ASC 2004, Banff, Canada, H. Leung (Ed.), ACTA Press, Anaheim - Calgary - Zürich (2003), pp. 283-289.

Appendix-I: generating rides and turns (up to TSP-10)

road	from_town	to_town	distance		road	from_town	to_town	distance
1	A	B	49		21	A	F	12
2	A	C	67		22	A	G	59
3	A	E	47		23	A	H	80
4	A	D	25		24	A	J	48
5	B	A	50		25	A	K	64
6	B	D	44		26	F	A	11
7	B	E	19		27	G	A	58
8	B	C	39		28	H	A	78
9	C	A	66		29	J	A	48
10	C	B	40		30	K	A	66
11	C	D	48		31	B	F	39
12	C	E	24		32	B	G	12
13	D	A	23		33	B	H	40
14	D	B	43		34	B	J	44
15	D	C	47		35	B	K	47
16	D	E	34		36	F	B	38
17	E	A	46		37	G	B	12
18	E	B	20		38	H	B	39
19	E	C	24		39	J	B	45
20	E	D	33		40	K	B	47

Table 1. Roads 1 – 40

road	from_town	to_town	distance		road	from_town	to_town	distance
41	C	F	62		66	F	E	39
42	C	G	32		67	G	E	18
43	C	H	19		68	H	E	32
44	C	J	29		69	J	E	27
45	C	K	13		70	K	E	29
46	F	C	62		71	F	G	49
47	G	C	33		72	F	H	72
48	H	C	19		73	F	J	47
49	J	C	28		74	F	K	61
50	K	C	13		75	G	F	49
51	D	F	27		76	G	H	29
52	D	G	50		77	G	J	45
53	D	H	62		78	G	K	41
54	D	J	23		79	H	F	71
55	D	K	42		80	H	G	29
56	F	D	27		81	H	J	46
57	G	D	50		82	H	K	31
58	H	D	63		83	J	F	47
59	J	D	23		84	J	G	45
60	K	D	42		85	J	H	46
61	E	F	39		86	J	K	20
62	E	G	18		87	K	F	61
63	E	H	32		88	K	G	42
64	E	J	28		89	K	H	30
65	E	K	29		90	K	J	20

Table 2. Roads 41 - 90

Queries for the generation of rides and turns

get road *its* “”, “| “, “1”.
get road *its* “”, “| “, “2”.
get road *its* “”, “| “, “3”.
etc. etc. …….
get road *its* “”, “| “, “10”.

Each *get* operation generates 90 roads followed by the specified integer expression. The empty string command “” presents the identifier of a road. The symbol “|” is produced in order to mark the preceding text (the identifier) as an integer expression. The result of this query is a report consisting of 900 lines containing a road identifier, followed by the text “| “ and the value for ‘ride *its* time_level’. This report is used as input for instances of “ride” (*type* ride = road, time_level). During the input process, the Xplain-DBMS offers the possibility to produce identifiers for the new instances automatically, which is applied here.

The number of roads is ***N*(N-1)***, and the number of rides per road is ***N***, so the number of lines in the resulting report is $N^2*(N-1)$. The time complexity of generating rides is $O(N^3)$.

Generating turn data-1 for town A:

get ride *its* "", "| ", road *its* from_town, "| ", time_level
where road *its* to_town = "A"
per time_level. /* sorting per time level.

Since the Xplain-DBMS does not contain secondary indices for attributes, all instances of 'ride' must be read; the reading costs of these get operation are proportional to the number of rides: $N^2*(N-1)$. Here the sorting costs (in RAM) are proportional to $^2log\{N^2*(N-1)\}$, but they can be ignored in comparison with the reading costs (disk-data): $N^2*(N-1) = O(N^3)$.

The number of roads going to the same town is $(N-1)$. There are N rides per road, so the number of instances (lines) in the resulting report containing rides to town A is: $N*(N-1)$.

Generating turn data-2 for town A:

extend ride *with* prevlevel = time_level –1.

get ride *its* "", "| ", road *its* to_town, "| ", prevlevel
where road *its* from_town = "A"
per prevlevel. /* sorting per previous time level.

The symbols "|" and " " have to mark and separate the integer expressions in the file produced by a query. The results of both queries were written to separate files and the result files were joined on the basis of the following condition: 'ride-1 *its* data-1 time-level = 'ride-2 *its* data-2 prevlevel'. Each report has $N*(N-1)$ lines, so the costs of joining two already sorted result files (reports) are proportional to $N*(N-1)$. The resulting join report was used as input for the generation of instances of 'help' using the following addition to our data model:

type help (i4) = first_ride (i5), start_town (a2), sec_ride (i5), finish_town (a2).

For each town the total costs of generating instances of 'help' are roughly proportional to N^3. Doing the same for all towns we find that the costs of generating help-data are $O(N^4)$.

In order to clean the subset with instances of 'help', we removed turns from ride XY to ride YX, in other words we removed turnarounds:

delete help *where* start_town = finish_town. The costs of this deletion: $O(N^4)$.

Producing the wanted subset of instances of 'turn':

get help *its* first_ride, "| ", sec_ride, "|".

The attributes 'help *its* first_ride' and 'help *its* sec_ride' are represented by integer expressions in the report file. Therefore, we mark and separate them with the text "| ", otherwise the report file cannot be used as input for values of the attributes 'turn *its* previous_ride' and 'turn *its* next_ride'. The results of this query were written to a report file and used as import data for instances of 'turn'. Apparently the total costs for generating all data on rides and turns are proportional to N^4.

Appendix-II: applying the *cascade* command to TSP-5

```
# APPROACH FOR A CYCLIC TOUR VIA 5 TOWNS STARTING IN TOWN-1 USING THE CASCADE-COMMAND:
value t1 = input(a2) "Enter town-1: ". value text1 = "town-1: ".  # READING USER INPUT
value text1. value t1.                          # PRINTING COMMANDS FOR text1 AND t1.
value t2 = input(a2) "Enter town-2: ". value text2 = "town-2: ". value text2. value t2.
value t3 = input(a2) "Enter town-3: ". value text3 = "town-3: ". value text3. value t3.
value t4 = input(a2) "Enter town-4: ". value text4 = "town-4: ". value text4. value t4.
value t5 = input(a2) "Enter town-5: ". value text5 = "town-5: ". value text5. value t5.
value inf = total road its distance.           # INITIALIZATION OF inf WITH THE HIGHEST
                                               # POSSIBLE LENGTH OF A CYCLE.
extend ride with distance = road its distance. # FASTER PROCESSING, SHORTER QUERY TEXT.
extend ride with fromtown = road its from_town. # IDEM.
extend ride with totown = road its to_town.     # IDEM.

extend ride with c =               # RIDES MUST GO BETWEEN TWO OF THE FIVE INPUT TOWNS.
((time_level = 1 and fromtown = t1
  and (totown = t2 or totown = t3 or totown = t4 or totown = t5)
 or (time_level = 5 and totown = t1
  and (fromtown = t2 or fromtown = t3 or fromtown = t4 or fromtown = t5))
 or (time_level > 1 and time_level < 5
  and (fromtown = t2 or fromtown = t3 or fromtown = t4 or fromtown = t5)
  and (totown = t2 or totown = t3 or totown = t4 or totown = t5)))).

extend turn with c = (previous_ride its c and next_ride its c).
                                  # TURNS MUST BE SUITABLE.
extend ride with sdistance = inf.# INITIALIZATION OF THE DISTANCE OF RIDES TO START.

update ride its sdistance = 0 where road its from_town = t1 and time_level = 1.
                                        # CORRECTION FOR THE SPECIFIED RIDES.
cascade ride its sdistance =            # CASCADING UPDATES OF ride its sdistance.
        min turn its previous_ride its sdistance + previous_ride its distance where c
        per next_ride.
extend ride with fdistance = inf. # INITIALIZATION OF THE DISTANCE OF RIDES TO FINISH.
update ride its fdistance = 0 where road its to_town = t1 and time_level = 5.
                                  # CORRECTION FOR THE SPECIFIED RIDES.
cascade ride its fdistance =      # CASCADING UPDATES OF ride its fdistance.
        min turn its next_ride its fdistance + next_ride its distance where c
        per previous_ride.
extend ride with totaldistance = sdistance + distance + fdistance.
value minimum = min ride its totaldistance.
extend ride with suitable =
             (totaldistance = minimum and fdistance < inf and sdistance < inf).
get ride its time_level,                # SHOW RESULTS.
             road its from_town, road its to_town,
             " sdist:", sdistance, " dist:", distance, " fdist:", fdistance,
             " totaldist:", totaldistance
    where suitable per time_level.      # ORDERING THE SELECTED RIDES PER time_level.
```

Results for a tour via the towns B, C, D, F, and G.

```
INPUT(1):     town-1: G    town-2: D    town-3: B    town-4: C    town-5: F
OUTPUT(1)     (the first column presents the id.'s of the retrieved rides,
               the second column presents time levels of retrieved rides):
   551     1 G     B      sdist:     0  dist:    12  fdist:   121  totaldist:   133
   113     2 B     D      sdist:    12  dist:    44  fdist:    77  totaldist:   133
   133     2 B     F      sdist:    12  dist:    39  fdist:    82  totaldist:   133
   314     3 D     F      sdist:    56  dist:    27  fdist:    50  totaldist:   133
   484     3 F     D      sdist:    51  dist:    27  fdist:    55  totaldist:   133
   285     4 D     B      sdist:    78  dist:    43  fdist:    12  totaldist:   133
   465     4 F     B      sdist:    83  dist:    38  fdist:    12  totaldist:   133
   146     5 B     G      sdist:   121  dist:    12  fdist:     0  totaldist:   133
Two incorrect cycles:  GBDFBG(133)   and   GBFDBG(133): B twice and C missed.
```

INPUT(2): town-1: F town-2: G town-3: D town-4: B town-5: C
OUTPUT(2):

481	1	F	D	sdist:	0	dist:	27	fdist:	130	totaldist:	157
293	2	D	C	sdist:	27	dist:	47	fdist:	83	totaldist:	157
234	3	C	G	sdist:	74	dist:	32	fdist:	51	totaldist:	157
555	4	G	B	sdist:	106	dist:	12	fdist:	39	totaldist:	157
136	5	B	F	sdist:	118	dist:	39	fdist:	0	totaldist:	157

A correct cycle of rides is found: FDCGBF (total distance: 157).

INPUT(3): town-1: D town-2: B town-3: C town-4: F town-5: G
OUTPUT(3):

311	1	D	F	sdist:	0	dist:	27	fdist:	126	totaldist:	153
463	2	F	B	sdist:	27	dist:	38	fdist:	88	totaldist:	153
144	3	B	G	sdist:	65	dist:	12	fdist:	76	totaldist:	153
595	4	G	F	sdist:	77	dist:	49	fdist:	27	totaldist:	153
486	5	F	D	sdist:	126	dist:	27	fdist:	0	totaldist:	153

An **incorrect** series of rides is found: D**F**BG**F**D(153): F twice and C missed.

INPUT(4): town-1: C town-2: F town-3: G town-4: D town-5: B
OUTPUT(4):

231	1	C	G	sdist:	0	dist:	32	fdist:	125	totaldist:	157
553	2	G	B	sdist:	32	dist:	12	fdist:	113	totaldist:	157
134	3	B	F	sdist:	44	dist:	39	fdist:	74	totaldist:	157
485	4	F	D	sdist:	83	dist:	27	fdist:	47	totaldist:	157
296	5	D	C	sdist:	110	dist:	47	fdist:	0	totaldist:	157

A correct cycle of rides is found: CGBFDC(total distance: 157).

INPUT(5): town-1: B town-2: C town-3: F town-4: G town-5: D
OUTPUT(5):

141	1	B	G	sdist:	0	dist:	12	fdist:	138	totaldist:	150
573	2	G	D	sdist:	12	dist:	50	fdist:	88	totaldist:	150
593	2	G	F	sdist:	12	dist:	49	fdist:	89	totaldist:	150
314	3	D	F	sdist:	62	dist:	27	fdist:	61	totaldist:	150
484	3	F	D	sdist:	61	dist:	27	fdist:	62	totaldist:	150
325	4	D	G	sdist:	88	dist:	50	fdist:	12	totaldist:	150
505	4	F	G	sdist:	89	dist:	49	fdist:	12	totaldist:	150
556	5	G	B	sdist:	138	dist:	12	fdist:	0	totaldist:	150

Two **incorrect** cycles: B**G**DF**G**B(150) and B**G**FD**G**B(150). G twice and C missed.

Results for a tour via the towns A, B, C, D and E

INPUT(6): town-1: A town-2: B town-3: C town-4: D town-5: E
OUTPUT(6):

21	1	A	D	sdist:	0	dist:	25	fdist:	118	totaldist:	143
283	2	D	B	sdist:	25	dist:	43	fdist:	75	totaldist:	143
124	3	B	E	sdist:	68	dist:	19	fdist:	56	totaldist:	143
395	4	E	D	sdist:	87	dist:	33	fdist:	23	totaldist:	143
276	5	D	A	sdist:	120	dist:	23	fdist:	0	totaldist:	143

A**D**BE**D**A(143): Incorrect because D is visited twice and C is missed.

INPUT(7): town-1: B town-2: C town-3: D town-4: E town-5: A
OUPUT(7):

121	1	B	E	sdist:	0	dist:	19	fdist:	123	totaldist:	142
393	2	E	D	sdist:	19	dist:	33	fdist:	90	totaldist:	142
274	3	D	A	sdist:	52	dist:	23	fdist:	67	totaldist:	142
35	4	A	E	sdist:	75	dist:	47	fdist:	20	totaldist:	142
376	5	E	B	sdist:	122	dist:	20	fdist:	0	totaldist:	142

B**E**DA**E**B(142): Incorrect because E is visited twice and C is missed.

```
INPUT(8):    town-1: C    town-2: D    town-3: E    town-4: A    town-5: B
OUTPUT(8):
   211     1 C     E      sdist:     0  dist:    24  fdist:   119  totaldist:   143
   393     2 E     D      sdist:    24  dist:    33  fdist:    86  totaldist:   143
   284     3 D     B      sdist:    57  dist:    43  fdist:    43  totaldist:   143
   125     4 B     E      sdist:   100  dist:    19  fdist:    24  totaldist:   143
   386     5 E     C      sdist:   119  dist:    24  fdist:     0  totaldist:   143
CEDBEC(143): Incorrect because E is visited twice and A is missed.

INPUT(9):    town-1: D    town-2: E    town-3: A    town-4: B    town-5: C
OUTPUT(9):
   301     1 D     E      sdist:     0  dist:    34  fdist:   116  totaldist:   150
   373     2 E     B      sdist:    34  dist:    20  fdist:    96  totaldist:   150
   383     2 E     C      sdist:    34  dist:    24  fdist:    92  totaldist:   150
   104     3 B     C      sdist:    54  dist:    39  fdist:    57  totaldist:   150
   194     3 C     B      sdist:    58  dist:    40  fdist:    52  totaldist:   150
   125     4 B     E      sdist:    98  dist:    19  fdist:    33  totaldist:   150
   215     4 C     E      sdist:    93  dist:    24  fdist:    33  totaldist:   150
   396     5 E     D      sdist:   117  dist:    33  fdist:     0  totaldist:   150
Two incorrect cycles found: DEBCED(150)  and DECBED(150). E visited twice and A missed.

INPUT(10):   town-1: E    town-2: A    town-3: B    town-4: C    town-5: D
OUTPUT(10):
   371     1 E     B      sdist:     0  dist:    20  fdist:   135  totaldist:   155
   113     2 B     D      sdist:    20  dist:    44  fdist:    91  totaldist:   155
   274     3 D     A      sdist:    64  dist:    23  fdist:    68  totaldist:   155
     5     4 A     B      sdist:    87  dist:    49  fdist:    19  totaldist:   155
   126     5 B     E      sdist:   136  dist:    19  fdist:     0  totaldist:   155
EBDABE(155): Incorrect because B is visited twice and C is missed.
```

Appendix-III: results of the greedy approach to TSP-5

```
####   GREEDY APPROACH FOR A CYCLIC TOUR VIA 5 SELECTED TOWNS STARTING IN town-1:
value t1 = input(a2) "Enter town-1: ". value text1 = "town-1: ". # t1:START AND FINISH.
value text1. value t1.                          # PRINT COMMANDS FOR text1 AND t1.
value t2 = input(a2) "Enter town-2: ". value text2 = "town-2: ". value text2. value t2.
value t3 = input(a2) "Enter town-3: ". value text3 = "town-3: ". value text3. value t3.
value t4 = input(a2) "Enter town-4: ". value text4 = "town-4: ". value text4. value t4.
value t5 = input(a2) "Enter town-5: ". value text5 = "town-5: ". value text5. value t5.
value inf = total road its distance.            # INITIALIZE inf WITH THE HIGHEST
                                                # POSSIBLE LENGTH OF A CYCLE.
extend ride with distance = road its distance.  # FASTER PROCESSING AND SHORTER QUERY.
extend ride with fromtown = road its from_town. # IDEM.
extend ride with totown = road its to_town.     # IDEM.
extend ride with c =                      # RIDES MUST GO BETWEEN TWO OF THE FIVE TOWNS.
((time_level = 1 and fromtown = t1
  and (totown = t2 or totown = t3 or totown = t4 or totown = t5)
 or (time_level = 5 and totown = t1
  and (fromtown = t2 or fromtown = t3 or fromtown = t4 or fromtown = t5))
 or (time_level > 1 and time_level < 5
  and (fromtown = t2 or fromtown = t3 or fromtown = t4 or fromtown = t5)
  and (totown = t2 or totown = t3 or totown = t4 or totown = t5)))).
extend turn with c =  (previous_ride its c and next_ride its c).
                                                        # TURNS MUST BE SUITABLE.
##### CALCULATIONS FOR THE SUITABLE RIDES HAVING time_level = 2:
extend turn with c1 = (previous_ride its time_level = 1 and c).
                          # TURNS FROM A FIRST TO A SECOND RIDE MUST BE CORRECT.
# INITIALIZATION OF ride its totaldistance TO THE STARTING TOWN t1:
extend ride with totaldist = inf.
update ride its totaldist = distance where time_level = 1 and c.
                                                    # CORRECTION OF SELECTED RIDES.
```

```
# CALCULATING THE SHORTEST RIDE WITH time_level = 1 TO NEXT RIDES WITH time_level = 2:
extend ride with min1 = min turn its previous_ride its totaldist where c1
                        per next_ride.
# SELECTION OF ride its town1 USING A SUITABLE PREVIOUS RIDE:
extend ride with town1 = some turn its previous_ride its fromtown where c1
                             and previous_ride its totaldist = next_ride its min1
                         per next_ride.
# SELECTION of ride its town2 USING A SUITABLE PREVIOUS RIDE:
extend ride with town2 = some turn its previous_ride its totown where c1
                           and previous_ride its totaldist = next_ride its min1
                         per next_ride.
# CALCULATION OF THE total distance FOR THE SELECTED RIDES:
update ride its totaldist = distance + min1 where time_level = 2 and c.

##### CALCULATIONS FOR THE SUITABLE RIDES HAVING time_level =3:
extend turn with c2 = (previous_ride its time_level = 2 and c).
extend ride with min2 = min turn its previous_ride its totaldist where c2
                        per next_ride.
extend ride with town3 = some turn its previous_ride its fromtown where c2
                             and previous_ride its totaldist = next_ride its min2
                         per next_ride.
extend ride with town4 = some turn its previous_ride its totown where c2
                             and previous_ride its totaldist = next_ride its min2
                         per next_ride.
extend ride with p5 = some turn its previous_ride its town1 where c2
                           and previous_ride its totaldist = next_ride its min2
                      per next_ride.
extend ride with p6 = some turn its previous_ride its town2 where c2
                           and previous_ride its totaldist = next_ride its min2
                      per next_ride.
update ride its town1 = p5 where time_level = 3 and c.
update ride its town2 = p6 where time_level = 3 and c.
update ride its totaldist = distance + min2 where time_level = 3 and c.

##### CALCULATIONS FOR THE SUITABLE RIDES HAVING time_level =4:
extend turn with c3 = (previous_ride its time_level = 3 and c
                       and next_ride its totown <> previous_ride its town2).
extend ride with min3 = min turn its previous_ride its totaldist where c3
                        per next_ride.
extend ride with town5 = some turn its previous_ride its fromtown where c3
                             and previous_ride its totaldist = next_ride its min3
                         per next_ride.
extend ride with town6 = some turn its previous_ride its totown where c3
                             and previous_ride its totaldist = next_ride its min3
                         per next_ride.
extend ride with p7 = some turn its previous_ride its town1 where c3
                           and previous_ride its totaldist = next_ride its min3
                      per next_ride.
extend ride with p8 = some turn its previous_ride its town2 where c3
                           and previous_ride its totaldist = next_ride its min3
                      per next_ride.
update ride its town1 = p7 where time_level = 4 and c.
update ride its town2 = p8 where time_level = 4 and c.
extend ride with p9 = some turn its previous_ride its town3 where c3
                           and previous_ride its totaldist = next_ride its min3
                      per next_ride.
extend ride with p10 = some turn its previous_ride its town4 where c3
                            and previous_ride its totaldist = next_ride its min3
                       per next_ride.
update ride its town3 = p9 where time_level= 4 and c.
update ride its town4 = p10 where time_level = 4 and c.
update ride its totaldist = distance + min3 where time_level = 4 and c.
```

```
##### CALCULATIONS FOR THE SUITABLE RIDES HAVING time_level =5:

extend turn with c4 = (previous_ride its time_level = 4 and c).
extend ride with min4 = min turn its previous_ride its totaldist where c4
                        per next_ride.
extend ride with town7 = some turn its previous_ride its fromtown where c4
                             and previous_ride its totaldist = next_ride its min4
                          per next_ride.
extend ride with town8 = some turn its previous_ride its totown where c4
                             and previous_ride its totaldist = next_ride its min4
                          per next_ride.
extend ride with p11 = some turn its previous_ride its town1 where c4
                            and previous_ride its totaldist = next_ride its min4
                        per next_ride.
extend ride with p12 = some turn its previous_ride its town2 where c4
                            and previous_ride its totaldist = next_ride its min4
                        per next_ride.
update ride its town1 = p11 where time_level = 5 and c.
update ride its town2 = p12 where time_level = 5 and c.
extend ride with p13 = some turn its previous_ride its town3 where c4
                            and previous_ride its totaldist = next_ride its min4
                        per next_ride.
extend ride with p14 = some turn its previous_ride its town4 where c4
                           and previous_ride its totaldist = next_ride its min4
                        per next_ride.
update ride its town3 = p13 where time_level = 5 and c.
update ride its town4 = p14 where time_level = 5 and c.
extend ride with p15 = some turn its previous_ride its town5 where c4
                            and previous_ride its totaldist = next_ride its min4
                        per next_ride.
extend ride with p16 = some turn its previous_ride its town6 where c4
                           and previous_ride its totaldist = next_ride its min4
                        per next_ride.

update ride its town5 = p15 where time_level = 5 and c.
update ride its town6 = p16 where time_level = 5 and c.
update ride its totaldist = distance + min4 where time_level = 5 and c.

newline.

value interresult =
    "Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5 ".
value interresult.

get ride its town1, town2, town3, town4, town5, town6, town7, town8, fromtown, totown,
      totaldist
   where time_level = 5 and town1 = t1 and totown = t1.

newline.

value minimum = min ride its totaldist
                    where time_level = 5 and town1 = t1 and totown = t1.
value finalresult =
    "Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5 ".
value finalresult.

get ride its town1, town2, town4, town6, town8, totown, totaldist
   where time_level = 5 and totown = t1 and town1 = t1 and totaldist = minimum.
```

Results for a cycle via B, C, D, F and G; comments between brackets

INPUT(1): town-1: G town-2: D town-3: B town-4: C town-5: F
OUTPUT(1):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
236 G B B F F D D C C G 157
326 G C C B B F F D D G 189
506 G B B C C D D F F G 175
Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5:
236 G B F D C G 157
{GBFDCG 157 equals the shortest cycle BFDCGB 157}

INPUT(2): town-1: F town-2: G town-3: D town-4: B town-5: C
OUTPUT(2):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
136 F D D C C G G B B F 157
226 F D D B B G G C C F 177
316 F B B G G C C D D F 158
596 F D D C C B B G G F 175
Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5:
136 F D C G B F 157
{FDCGBF 157 equals the shortest cycle BFDCGB 157}

INPUT(3): town-1: D town-2: B town-3: C town-4: F town-5: G
OUTPUT(3):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
206 D F F B B G G C C D 158
486 D B B G G C C F F D 177
576 D B B F F C C G G D 226
Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5:
206 D F B G C D 158
{equals BGCDFB 158, reverse of the shortest cycle BFDCGB 157}

INPUT(4): town-1: C town-2: F town-3: G town-4: D town-5: B
OUTPUT(4):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
106 C D D F F G G B B C 175
296 C G G B B F F D D C 157
476 C G G B B D D F F C 177
566 C D D F F B B G G C 158
Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5:
296 C G B F D C 157
{CGBFDC 157 equals the shortest cycle BFDCGB 157}

INPUT(5): town-1: B town-2: C town-3: F town-4: G town-5: D
OUTPUT(5):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
196 B G G F F D D C C B 175
286 B G G C C F F D D B 177
466 B G G C C D D F F B 158
Shortest cycle with start and finish in town-1 via the towns -2, -3, -4, and -5:
466 B G C D F B 158
{equals BGCDFB 158, reverse of the shortest cycle BFDCGB 157}

Results for a cycle via A, B, C, D and E

INPUT(6): town-1: A town-2: B town-3: C town-4: D town-5: E
OUTPUT(6):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
96 A D D C C E E B B A 166
186 A D D B B E E C C A 177
276 A E E B B C C D D A 177
Shortest cycle with start and finish in town-1 via the towns-2, -3, -4, and -5:
96 A D C E B A 166
{equals ADCEBA 166, reverse of the shortest cycle ABECDA 163}

INPUT(7): town-1: B town-2: C town-3: D town-4: E town-5: A
OUTPUT(7):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
6 B E E C C D D A A B 163
196 B A A D D E E C C B 173
286 B E E C C A A D D B 177
Shortest cycle with start and finish in town-1 via the towns-2, -3, -4, and -5:
6 B E C D A B 163
{BECDAB equals the shortest cycle ABECDA 163}

INPUT(8): town-1: C town-2: D town-3: E town-4: A town-5: B
OUTPUT(8):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
16 C E E B B D D A A C 178
106 C E E D D A A B B C 168
296 C E E B B A A D D C 166
Shortest cycle with start and finish in town-1 via the towns-2, -3, -4, and -5:
296 C E B A D C 166
{CEBADC equals ADCEBA 166, reverse of the shortest cycle ABECDA 163}

INPUT(9): town-1: D town-2: E town-3: A town-4: B town-5: C
OUTPUT(9):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
26 D E E C C B B A A D 173
116 D E E C C A A B B D 217
206 D A A B B E E C C D 163
Shortest cycle with start and finish in town-1 via the towns-2, -3, -4, and -5:
206 D A B E C D 163
{DABECD equals the shortest cycle ABECDA 163}

INPUT(10): town-1: E town-2: A town-3: B town-4: C town-5: D
OUTPUT(10):
Series of rides with start and finish in town-1 via the towns -2, -3, -4 and -5:
36 E B B C C D D A A E 177
126 E D D A A C C B B E 182
216 E B B A A D D C C E 166
306 E C C B B A A D D E 173
Shortest cycle with start and finish in town-1 via the towns-2, -3, -4, and -5:
216 E B A D C E 166
{EBADCE equals ADCEBA 166, reverse of the shortest cycle ABECDA 163}

Appendix-IV: Time complexity of query language approaches to asymmetric TSP

Complying with the definition, each pair of towns is connected by two single-direction roads with reverse directions and ***different*** lengths [1, 5, 7, and 8]. Then the number of single-direction roads is ***N*(N-1)***. A round tour through ***N*** towns consists of ***N*** successive rides and each of these ***N*** towns is visited only once. Using rides and turns, a suitable number of time levels must be chosen. Referring to figure 3, it is clear that we have to define ***N*** time levels: ***N*** rides for each road. The number of single-direction roads is ***N*(N-1)***, the total number of rides is ***N*N*(N-1)***. However, in our database the rides at the highest time level are not followed by other rides; therefore the number of rides that can be followed by other rides is ***(N-1)*N*(N-1)***.

A ride XY can be succeeded by ***(N-1)*** rides starting in town Y, but the reverse ride YX is not suitable. Therefore, turnarounds are not present in our database created for TSP-cases up to ten towns. Consequently for each of the rides that can be followed by another ride the number of possible successive rides is ***(N-2)***. Thus the number of turns (arcs) suitable for a cyclic tour over ***N*** towns to be registered is not $(N-1)*N*(N-1)^2$, but $(N-2)*N*(N-1)^2$.

For the time complexity of our two query language approaches, we consider both data generation and query processing.

Costs of data generation

As explained during the discussion on data generation, the total costs of data generation are roughly proportional to N^4, where ***N*** is the number of towns.

Costs of the approach using the cascading update

For TSP-5 the number of set operations (*extend*, *update*, *get*) on instances of 'ride' is 12. The number of rides is ***N*N*(N-1)***, so the costs are $O\{N^3\}$. The number of set operations on instances of 'turn' is 1. The cost of this operation on 'turn' is proportional to the number of turns $(N-2)*N*(N-1)^2$: $O\{N^4\}$. We used two *cascade* commands. The costs of this operation is proportional to ***d*A*** [16], where ***d*** is the depth of the graph and ***A*** is the number of arcs. In our case ***d*** equals the number of time levels ***N***, whereas ***A*** equals the number of turns: $(N-2)*N*(N-1)^2$. As a result, the costs of the two cascade commands are proportional to N^5. Therefore, the overall time complexity of the query using the *cascade* command is: $O\{N^5\}$.

Costs of the greedy approach

The greedy approach applies a step by step approach. Although the number of operations within the type 'ride' increases within each following step, a rough estimation is that in each step the number of operations within the type 'ride' is proportional to the number of rides: ***N*N*(N-1),*** which leads to a time complexity $O\{N^3\}$. In each step we applied one *extend* operation where the set of turns is involved, so the costs are proportional to the number of turns: $O\{N^4\}$.

The number of steps is ***N-1***, consequently, the roughly estimated overall time complexity of this greedy approach is: $O\{N^5\}$.