

GraphTheory

A substantial effort was put into Graph Theory for Maple 2023, including improved ability to solve traveling salesman problems, support for multigraphs, new commands for graph computation, and advances in visualization.

> *with(GraphTheory)* :

[Traveling Salesman](#)

[Multigraph support](#)

[Graph products](#)

[Additions to SpecialGraphs](#)

Traveling Salesman

The [TravelingSalesman](#) command now makes use of [Concorde](#), a well-known library implementing highly efficient heuristics for solving instances of the traveling salesman problem. This addition considerably increases the size of problems which `TravelingSalesman` is able to handle.

Example: Using TravelingSalesman with vertexpositions

In the following example, we import a dataset of the 100 largest cities in the continent of Africa (including the island of Madagascar) and ask `TravelingSalesman` to find a minimal tour of them.

> *AfricanCities := Import("example/AfricanCities100.tsv", base = datadir, output = Matrix, skiplines = 1)*

```
AfricanCities :=
```

1	"Cairo"	"Egypt"	22183200	30.044444	31.235833
2	"Kinshasa"	"Democratic Republic of the Congo"	16315534	-4.325	15.322222
3	"Lagos"	"Nigeria"	15387639	6.455027	3.384082
4	"Giza"	"Egypt"	9200000	29.987	31.2118
5	"Luanda"	"Angola"	8952496	-8.838333	13.234444
6	"Dar es Salaam"	"Tanzania"	7404689	-6.816111	39.280278
7	"Khartoum"	"Sudan"	6160327	15.500556	32.56
8	"Johannesburg"	"South Africa"	6065354	-26.204444	28.045556
9	"Abidjan"	"Côte d'Ivoire"	5515790	5.316667	-4.033333
10	"Alexandria"	"Egypt"	5483605	31.1975	29.8925
⋮	⋮	⋮	⋮	⋮	⋮

100 × 6 Matrix

```
> G := CompleteGraph( convert(AfricanCities[ ..., 2], list), vertexpositions = AfricanCities[ ..., [6, 5]])
```

G := Graph 1: an undirected graph with 100 vertices and 4950 edge(s)

With the new vertexpositions option, TravelingSalesman uses edge weights computed from the geometric distance between vertices in the graph layout specified previously when CompleteGraph was called. The new startvertex option allows a particular starting vertex to be specified.

```
> W, T := TravelingSalesman( G, vertexpositions, startvertex = "Cairo")
```

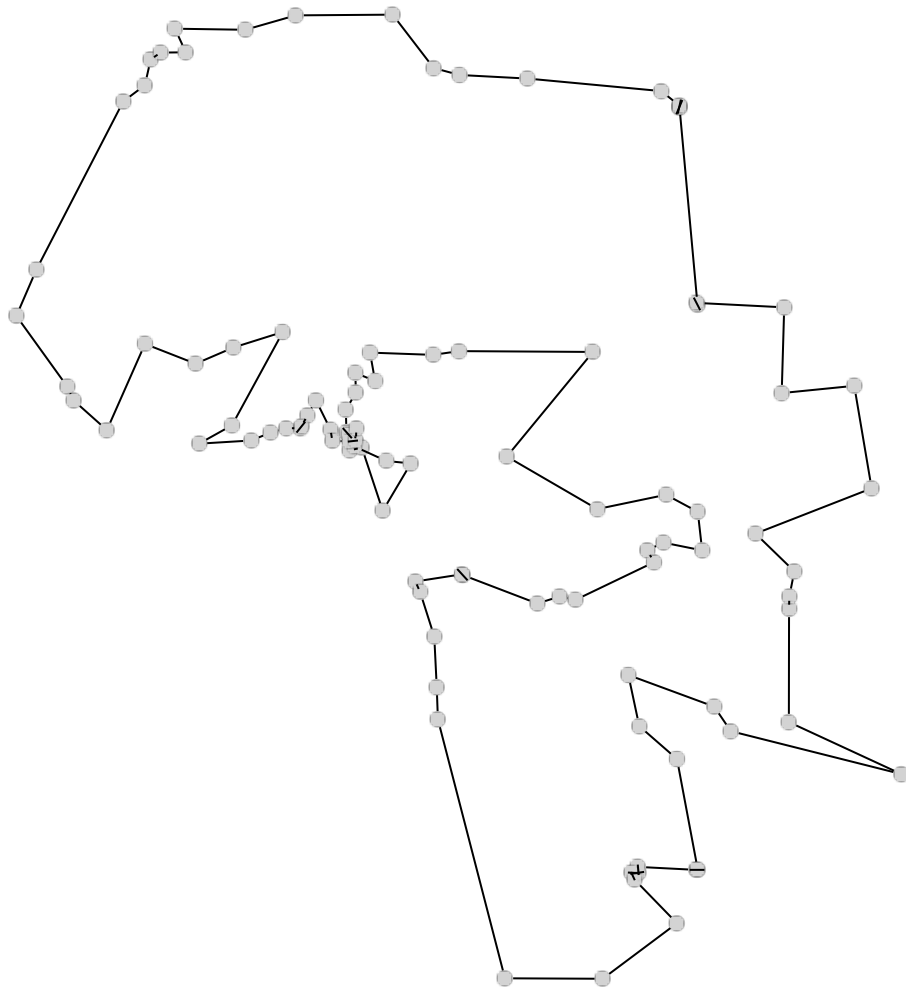
```
W, T := 421.032920193835, ["Cairo", "Alexandria", "Benghazi", "Misratah", "Tripoli", "Tunis",
"Algiers", "Oran", "Tangier", "Fez", "Rabat", "Casablanca", "Marrakesh", "Agadir", "Nouakchott",
"Dakar", "Conakry", "Freetown", "Monrovia", "Bamako", "Bobo Dioulasso", "Ouagadougou",
"Niamey", "Ilorin", "Ibadan", "Kumasi", "Abidjan", "Accra", "Lomé", "Abomey-Calavi", "Lagos",
"Ikorodu", "Benin City", "Warri", "Owerri", "Umuahia", "Onitsha", "Nnewi", "Port Harcourt",
"Yaoundé", "Douala", "Libreville", "Uyo", "Aba", "Enugu", "Lokoja", "Abuja", "Kaduna", "Jos",
"Kano", "Maiduguri", "N'Djamena", "Nyala", "Bangui", "Kisangani", "Bunia", "Kampala",
"Mwanza", "Kigali", "Bujumbura", "Bukavu", "Mbuji-Mayi", "Kananga", "Tshikapa", "Kinshasa",
"Brazzaville", "Pointe-Noire", "Cabinda", "Luanda", "Benguela", "Lubango", "Cape Town",
"Gqeberha (Port Elizabeth)", "Durban (eThekweni)", "Vereeniging (Emfuleni)", "West Rand",
"Johannesburg", "East Rand (Ekurhuleni)", "Pretoria (Tshwane)", "Matola", "Maputo", "Harare",
"Lusaka", "Lubumbashi", "Lilongwe", "Blantyre", "Antananarivo", "Nampula", "Dar es Salaam",
"Zanzibar", "Mombasa", "Nairobi", "Mogadishu", "Hargeisa", "Addis Ababa", "Asmara",
"Omdurman", "Khartoum", "Giza", "Shubra el-Kheima", "Cairo"]
```

We will illustrate the tour by constructing a subgraph consisting only of the edges included in the optimal tour across G.

> $TG := \text{Subgraph}(G, \text{Trail}(T))$;

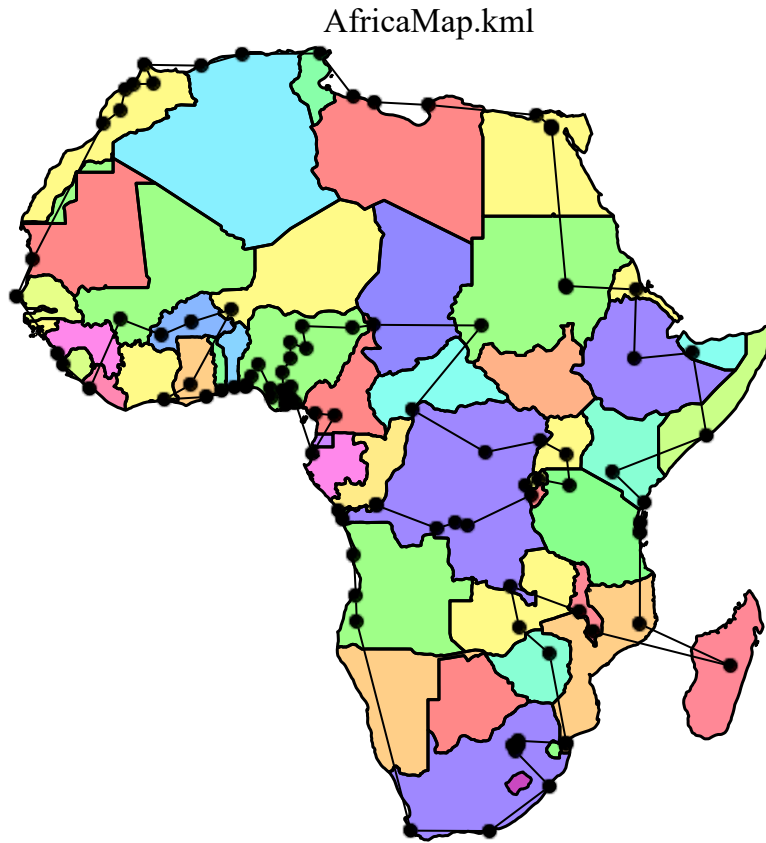
$TG := \text{Graph 2: an undirected graph with 100 vertices and 100 edge(s)}$

> DrawGraph(TG)



To better visualize the tour we can combine this with a country map of Africa.

```
> plots:-display(Import("example/AfricaMap.kml", base = datadir), DrawGraph(TG, stylesheet  
= [vertexcolor = black]))
```



Using TravelingSalesman with an arbitrary weight matrix

The previous example demonstrates the use of TravelingSalesman with edge weights corresponding directly to geometric distances between vertices. In many instances of the traveling salesman problem, the weights do not correspond so directly to the geometry.

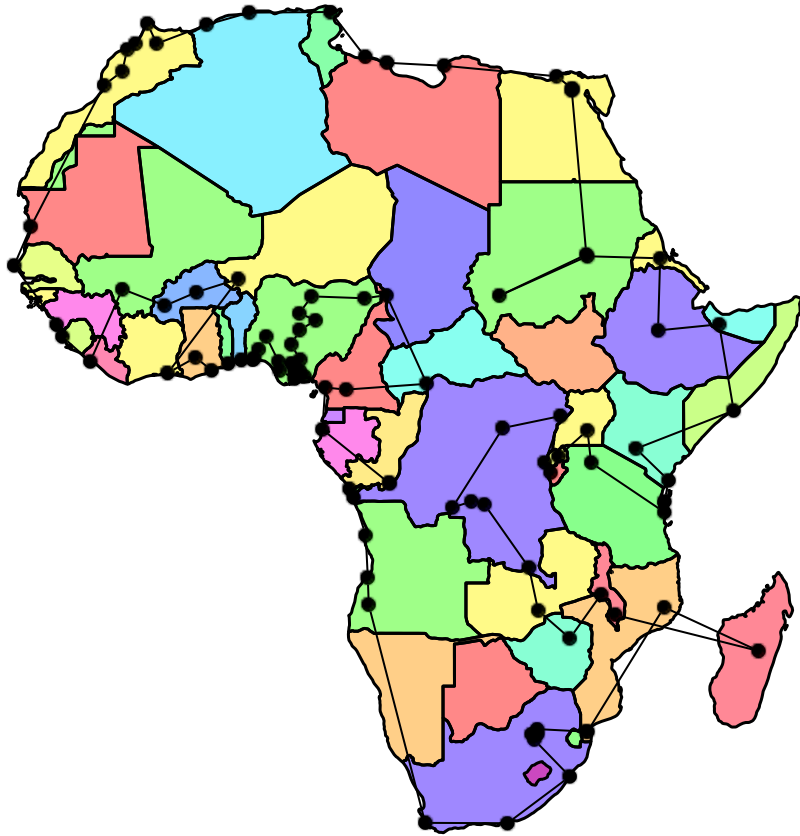
Fortunately, TravelingSalesman can also compute a tour when given an arbitrary weight matrix. To illustrate this, let us extend the previous example of a tour of 100 African cities to incorporate the idea that there might be some increased cost associated with traversing an international border.

> $TG2 := \text{Subgraph}(G, \text{Trail}(T2))$

$TG2 := \text{Graph } 3: \text{ an undirected graph with } 100 \text{ vertices and } 100 \text{ edge(s)}$

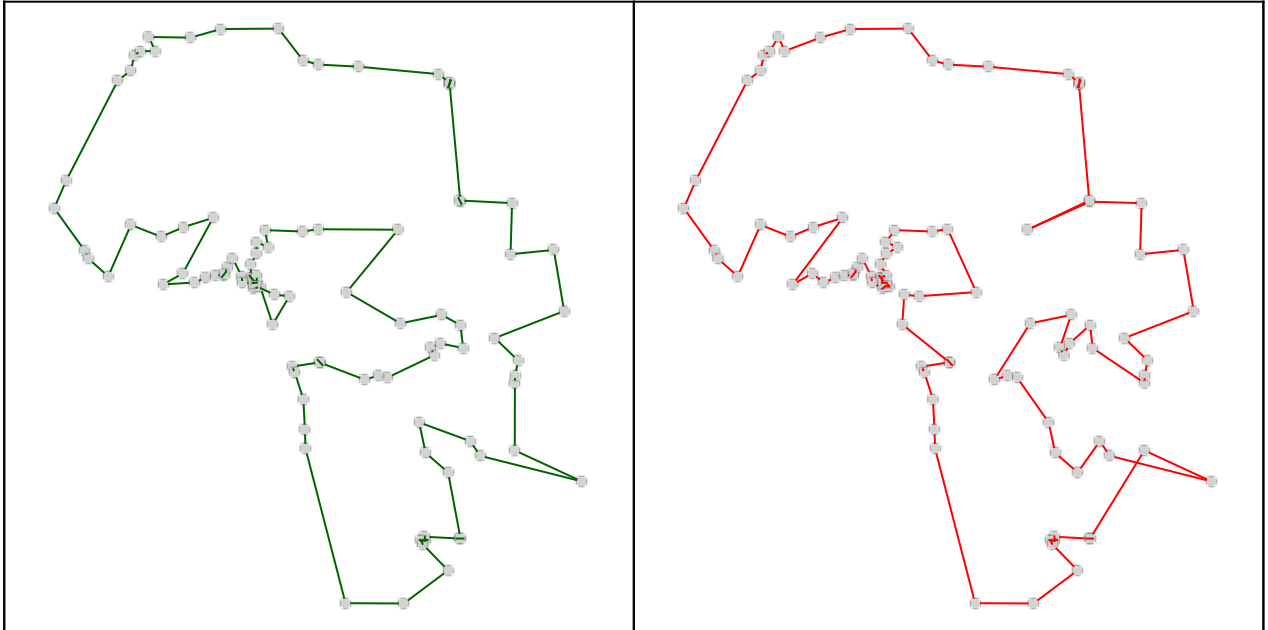
This new tour is similar to the previous one in many respects, but notably does not visit Sudan or the Democratic Republic of the Congo twice as the first one did, illustrating the effect of our increased edge weights.

> *plots:-display(Import("example/AfricaMap.kml", base = datadir, title = none),
DrawGraph(TG2, stylesheet = [vertexcolor = black]))*



Finally we can draw the original tour (in green) and the new tour (in red) to see the effect of the altered weights.

```
> plots:-display( (DrawGraph(TG, stylesheet = [edgecolor = "darkgreen"]) | DrawGraph(TG2,
  stylesheet = [edgecolor = red])) )
```



Multigraph support

GraphTheory now supports multigraphs. That is, graphs in which there may be multiple edges between the same pair of vertices.

```
> MG := Graph( 4, [ [ 0 2 0 0 ]
  [ 2 0 1 0 ]
  [ 0 1 0 2 ]
  [ 0 0 2 0 ] ], multigraph)
```

MG := Graph 3: an undirected multigraph with 4 vertices and 5 edge(s)

The new command [IsMultigraph](#) tests whether a graph is a multigraph.

```
> IsMultigraph(MG)
```

true

Many other commands have been updated to support multigraphs.

The [Edges](#) command for this graph returns a list, not a set, and repeats the edge between vertices 3 and 4 twice.

```
> Edges(MG)
```

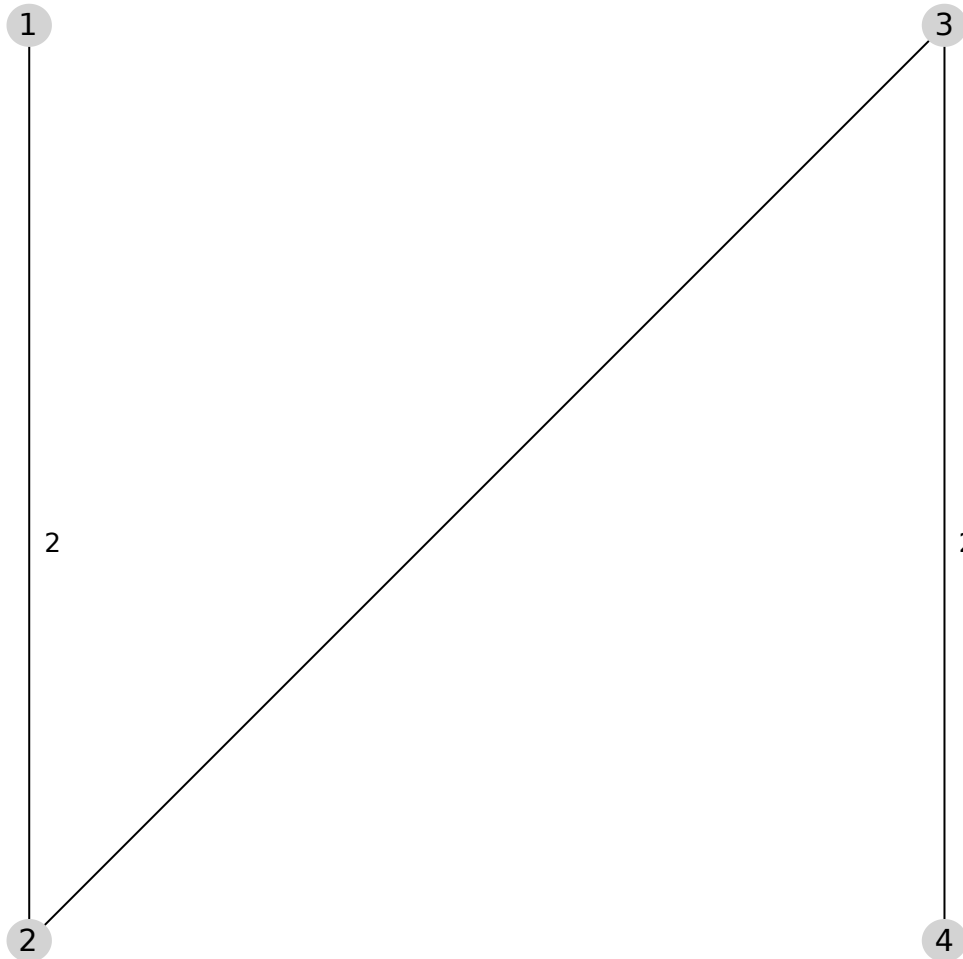
[{1, 2}, {1, 2}, {2, 3}, {3, 4}, {3, 4}]

> $EdgeMultiplicity(MG, \{1, 2\})$

2

Graph visualization commands such as `DrawGraph` will draw an integer weight on edges for which the edge multiplicity is greater than 1.

> $DrawGraph(MG)$



Graph products

A *graph product* is a binary operation on graphs which takes two graphs G_1 and G_2 and produces a graph H with the following properties:

- The vertex set of H is the Cartesian product $V(G_1) \times V(G_2)$ where $V(G_1)$ and $V(G_2)$ are the vertex sets of G_1 and G_2 , respectively.

$V(G_1)$

$V(G_2)$

- Two vertices (u_1, v_1) and (u_2, v_2) of H are connected by an edge, iff some condition about $u_1, v_1 \in G_1$ and $u_2, v_2 \in G_2$ is fulfilled.

Many different graph products have been defined which differ in the condition imposed on the edges. To the existing [CartesianProduct](#) and [TensorProduct](#) commands, the following new graph products have been added:

Name	Edge Condition
ConormalProduct	u_1 and v_1 share an edge in G_1 or u_2 and v_2 share an edge in G_2
LexicographicProduct	u_1 and v_1 share an edge in G_1 , or $u_1 = v_1$ in G_1 and u_2 and v_2 share an edge in G_2
ModularProduct	u_1 and v_1 share an edge in G_1 and u_2 and v_2 share an edge in G_2 , or u_1 and v_1 do not share an edge in G_1 and u_2 and v_2 do not share an edge in G_2
StrongProduct	$u_1 = v_1$ in G_1 and u_2 and v_2 share an edge in G_2 , or u_1 and v_1 share an edge in G_1 and $u_2 = v_2$ in G_2 or u_1 and v_1 share an edge in G_1 and u_2 and v_2 share an edge in G_2

> $C := \text{CycleGraph}(3)$

$C := \text{Graph 4: an undirected graph with 3 vertices and 3 edge(s)}$

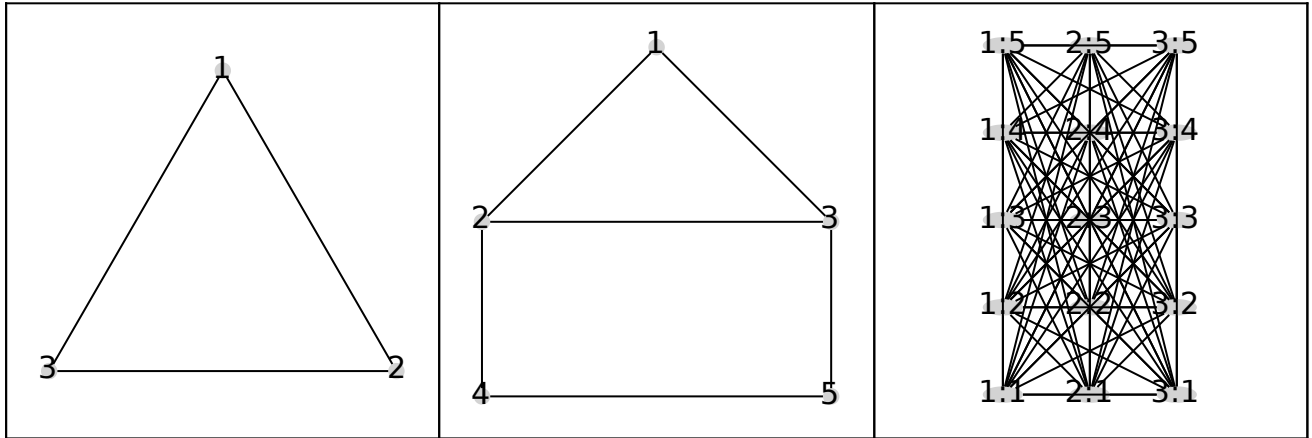
> $H := \text{SpecialGraphs:-HouseGraph}()$

$H := \text{Graph 5: an undirected graph with 5 vertices and 6 edge(s)}$

> $LP := \text{LexicographicProduct}(C, H)$

$LP := \text{Graph 6: an undirected graph with 15 vertices and 93 edge(s)}$

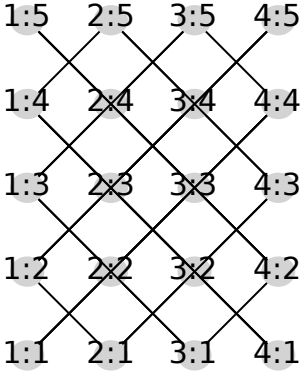

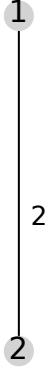
> plots:-display(DrawGraph~((C|H|LP)))

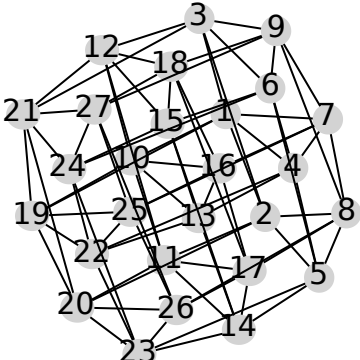
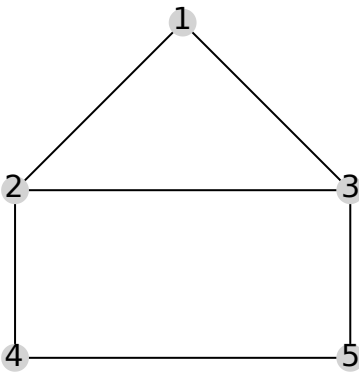


Additions to SpecialGraphs

Maple 2023 provides support for 6 additional Special Graphs, bringing the total to 119.

> *with(SpecialGraphs)* :

Bishops's Graph	Bouquet Graph	Dipole Graph
<pre data-bbox="245 499 625 982"> > BG46 := BishopsGraph(4, 5) BG46 := Graph 7: an undirected graph with 20 vertices and 40 edge(s) > DrawGraph(BG46, stylesheet = [vertexpadding = 12]) </pre> 	<pre data-bbox="659 499 1057 934"> > BG := BouquetGraph(5) BG := Graph 8: an undirected multigraph with 1 vertex, no edges, and 5 self-loops > DrawGraph(BG, stylesheet = [vertexpadding = 12]) </pre> 	<pre data-bbox="1081 499 1455 934"> > DG := DipoleGraph(2) DG := Graph 9: an undirected multigraph with 2 vertices and 2 edge(s) > DrawGraph(DG, stylesheet = [vertexpadding = 12]) </pre> 

Hamming Graph	House Graph	Windmill Graph
<p>> HG33 := HammingGraph(3, 3) HG33 := Graph 10: an undirected graph with 27 vertices and 81 edge(s)</p> <p>> DrawGraph(HG33, style = spring, stylesheet = [vertexpadding = 12])</p> 	<p>> HG := HouseGraph() HG := Graph 11: an undirected graph with 5 vertices and 6 edge(s)</p> <p>> DrawGraph(HG, stylesheet = [vertexpadding = 10])</p> 	<p>> WG := WindmillGraph(3, 7) WG := Graph 12: an undirected graph with 15 vertices and 21 edge(s)</p> <p>> DrawGraph(WG, stylesheet = [vertexpadding = 10])</p> 