GYutsis: heuristic based calculation of general recoupling coefficients

D. Van Dyck, V. Fack

Research Group Combinatorial Algorithms and Algorithmic Graph Theory,
URL: `http://caagt.rug.ac.be/`,
Department of Applied Mathematics & Computer Science,
Krijgslaan 281–S9, 9000 Ghent, Belgium

PACS: 02.70; 03.65; 31.15

Abstract

General angular momentum recoupling coefficients can be expressed as a summation formula over products of 6-j coefficients. Yutsis, Levinson and Vanagas developed graphical techniques for representing the general recoupling coefficient as a cubic graph and they describe a set of reduction rules allowing a stepwise generation of the corresponding summation formula. This paper is a follow up to [16] where we described a heuristic algorithm based on these techniques. In this article we separate the heuristic from the algorithm and describe some new heuristic approaches which can be plugged into the generic algorithm. We show that these new heuristics lead to good results: in many cases we get a more efficient summation formula than our previous approach, in particular for problems of higher order. In addition the new features and the use of our program `GYutsis`, which implements these techniques, is described both for end users and application programmers.

Key words: Angular momentum; General recoupling coefficient; Yutsis graph; Reduction rules; Cyclic structure; Heuristic.

PROGRAM SUMMARY

Title of program: CycleCostAlgorithm, GYutsis
Catalogue number:
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue). Users may obtain the program also by downloading either the compressed tar file `gyutsis.tgz` (for Unix and Linux) or the zip file `gyutsis.zip` (for Windows) from our website (`http://caagt.rug.ac.be/yutsis/`). An applet version of the program is also available on our website and can be run in a web browser from the URL `http://caagt.rug.ac.be/yutsis/GYutsisApplet.html`.
Licensing provisions: none
Computers for which the program is designed: any computer with Sun's Java Runtime Environment 1.4 or higher installed.
Programming language used: Java 1.2 (Compiler: Sun's SDK 1.4.0)
No. of lines in program: approximately 9400
Nature of physical problem: A general recoupling coefficient for an arbitrary number of (integer or half-integer) angular momenta can be expressed as a formula consisting of products of 6-j coefficients summed over a certain number of variables. Such a formula can be generated using the program `GYutsis` (with a graphical user front end) or `CycleCostAlgorithm` (with a text-mode user front end).
Method of solution: Using the graphical techniques of Yutsis, Levinson and Vanagas [18] a summation formula for a general recoupling coefficient is obtained by representing the coefficient as a Yutsis graph and by performing a selection of reduction rules valid for such graphs. Each reduction rule contributes to the final summation formula by a numerical factor or by an additional summation variable. Whereas an optimal summation formula (i.e. with a minimum number of summation variables) is hard to obtain, we present here some new heuristic approaches for selecting an edge from a k-cycle in order to transform it into an $(k-1)$-cycle ($k > 3$) in such a way that a 'good' summation formula is obtained.
Typical running time: From instantaneously for the typical problems to 30 seconds for the heaviest problems on a Pentium II-350 Linux-system with 256 MB RAM.

LONG WRITE-UP

1 Introduction

In various fields of theoretical physics, the quantum mechanical description of many-particle processes often requires an explicit transformation of the angular momenta of the subsystems among different coupling schemes. Such transformations are described by *general recoupling coefficients* and arise mostly in atomic and nuclear structure and scattering calculations [3]. Several algorithms have been described to generate a summation formula expressing the recoupling coefficient as a multiple sum over products of Wigner 6-j symbols multiplied by phase factors and square root factors [2,4,8,13,16]. The aim is to find an optimal summation formula, i.e. with a minimum number of summation variables and Wigner 6-j symbols.

The best algorithms at present are based on techniques developed by Yutsis, Levinson and Vanagas [18] and manipulate a graphical representation of the recoupling coefficient called a Yutsis graph. Reduction rules are defined for these graphs, which allow a stepwise transformation of the graph by reduction and removal of cycles. Each reduction step contributes part of the final summation formula.

As the optimality of the generated formula depends strongly on which rule is selected in each step and on which cycle of the graph it is applied [4,16], we present here some heuristic approaches for making that choice by examining the cyclic structure of the graph. As will be clear from our experiments, these approaches yield significantly better formulae, in particular for problems of higher order.

The paper is organized as follows. Section 2 recalls some notions from the theory of angular momentum theory and Yutsis graphs. Section 3 describes algorithm `CCA` and analyzes its asymptotic behaviour. In Section 4 we shortly recall the heuristic `EC` based on edge costs introduced in [16], investigate the performance of the heuristic algorithm `CCA-EC` and show how the technique of limited discrepancy search (LDS) [11] can be used to obtain even better formulae, which also gives rise to the improved heuristics described in Section 5. The use and implementation details of the program itself are described in Section 6. Finally Section 7 compares and discusses the results of the improved heuristics versus the heuristic used in [16].

2 Graphical representation of recoupling coefficients

This section summarizes some notions from the quantum theory of angular momenta, showing how a Yutsis graph is constructed for a given recoupling coefficient and which reduction rules are used in this paper. For the general theory of Yutsis graphs we refer to [18] and [3].

Consider a general recoupling coefficient of $n+1$ integer and half integer angular momenta. With each label in the recoupling coefficient an edge in the graph is associated and with each coupling a node is associated, resulting in a cubic graph with $2n$ nodes and $3n$ edges. The nodes representing the coupling of the left-hand side of the recoupling coefficient get a '$-$'-sign, those on the right-hand side get a '$+$'-sign. The edges corresponding to the compounded angular momenta on the left-hand side are directed away from the node while the edges representing the resultant are directed towards the node. The direction of the edges corresponding to the left-hand side are the reverse of those corresponding to the right-hand side.

The sign of a node where angular momenta j_1, j_2 and j_3 meet can be inverted by multiplying the value the graph represents by $(-1)^{j_1+j_2+j_3}$. A change of direction of an edge with label j results in a multiplication by $(-1)^{2j}$. The transformation coefficient then equals the j coefficient represented by the diagram multiplied by (see [18], equations (22.1) and (22.2)):

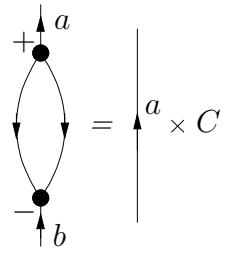
$$(-1)^{2(J+\sum_{i=1}^{n-1} b_i+S)}[\prod_{i=1}^{n-1}(2a_i+1)(2b_i+1)]^{1/2}\,,$$

with S the sum of all 'first' coupled angular momenta, $n+1$ the number of angular momenta, a_i the intermediate angular momenta on the left side, b_i the intermediate angular momenta on the right side, and J the total angular momentum.

Once the graph is generated, it can be simplified with the help of the reduction rules developed by Yutsis, Levinson and Vanagas [18]. Here we use only three rules: reduction of bubbles, reduction of triangles and the interchange operation. Figure 1 shows a graphical interpretation of these rules. For our purposes it is important to note that the reduction of a triangle can be seen as an interchange followed by the removal of a bubble. Using these rules the reduction algorithms iteratively eliminate cycles from the Yutsis graph, until the graph is simplified to a so-called "triangular delta", i.e. a graph consisting of two nodes connected by three parallel edges.

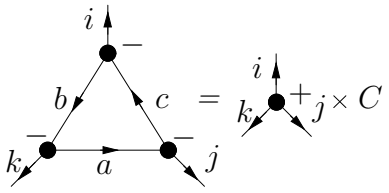
From an algorithmic point of view we are only interested in the complexity of the formula. Since the difference between the number of summations and the

Rule I: reduction of a 2-cycle (bubble)



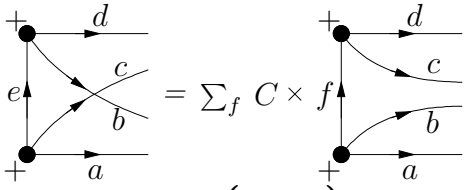
with $C = (2a+1)^{-1}\delta(a,b)$.

Rule II: reduction of a 3-cycle (triangle)



with $C = \left\{ \begin{array}{ccc} i & j & k \\ a & b & c \end{array} \right\}$.

Rule III: interchange



with $C = (-1)^{b+c+e+f}(2f+1) \left\{ \begin{array}{ccc} a & b & f \\ d & c & e \end{array} \right\}$.

Fig. 1. Graphical reduction rules for Yutsis graphs.

number of 6-j coefficients is constant, the number of 6-j coefficients suffices as measure for the complexity of the formula. With this idea in mind we define the cost of a reduction rule as the number of 6-j coefficients the rule yields. This equals the number of interchanges needed to decompose the rule to a sequence of interchanges followed by the bubble rule. With the same reasoning we can drop the signs of the nodes and the direction of the edges, since they only contribute in phase and weight factors, not influencing the complexity of the generated summation formula.

3 Algorithm for reducing Yutsis graphs (CCA)

3.1 Algorithm description

Like most other algorithms the algorithm CCA iteratively eliminates cycles from the Yutsis graph until a triangular delta is obtained, in each reduction step reducing a smallest cycle in length. When several choices are possible, heuristics are used to select a 'best choice'. In this section we describe the generic algorithm CCA, without specifying the details of the heuristics. The heuristics used are described in Sections 4 and 5. This algorithm is a generalization of the algorithm described in [16], in which the heuristic, used to select an operation when no triangles or bubbles are present, is separated from the algorithm. We refer to the heuristic based on edge costs used in [16] as EC and to the algorithm described in [16] as CCA-EC.

Examining the cyclic structure of the graph has been the bottleneck for all the older algorithms. For cases with girth [1] greater than 4, these algorithms spend almost all the time needed to generate a summation formula looking for cycles. The same principle is used by all of them: first look for cycles of length 3, then 4, and so on, each step taking more time. Our algorithm uses the algorithm of Vismara [17] to generate all *relevant cycles*. A cycle is called relevant if it belongs to at least one minimum cycle basis. The set of all relevant cycles always contains all the girth cycles (see [17] for more details).

First the algorithm searches for bubbles and triangles and removes any bubble or triangle immediately. If no bubbles or triangles are present, we generate all the relevant cycles of the cubic graph using Vismara's algorithm and we select an edge on which to apply an interchange based on a heuristic examining the cyclic structure. A pseudocode formulation of the algorithm is given in Figure 2.

It is interesting to note that reducing a triangle only makes relevant cycles smaller. Indeed, since every relevant cycle is composed of 2 shortest paths of equal length (for an even cycle) or 2 shortest paths of equal length and an edge (for an odd cycle) [17], another relevant cycle can have at most one edge in common with the triangle. This edge must be part of one of the defining paths of the cycle or it must be the additional edge in the construction of an odd cycle. In both cases the reduction of the triangle reduces the other cycle one unit in length. If no edge is shared the length of the cycle is not influenced.

[1] The *girth* of a graph is the size of its smallest cycle. A *girth cycle* is a cycle whose length is equal to the girth.

```
CCA (YutsisGraph Y)
while Y != triangular delta do
  if Y has bubble then
    Format and remove arbitrary bubble
  else if Y has triangle then
    Format and remove arbitrary triangle
  else
    Generate all relevant cycles (Vismara's algorithm)
    Use heuristic to select the best cycle and its best edge
    Format and interchange best edge out of the best cycle
  end if
end while
return formula
```

Fig. 2. Generic algorithm to generate a summation formula for a general recoupling coefficient from the corresponding Yutsis graph.

3.2 About the number of paths and cycles

In order to analyze the time complexity of the algorithm, we have to make an assumption on the number of shortest paths and cycles in a Yutsis graph. Let Y be a Yutsis graph with nodeset V, with $|V| = 2n$, and edgeset E, with $|E| = 3n$. We call a path *cyclic canonical* if it starts with the smallest node in some ordering.

As in [16] we assume that the number of shortest paths between two arbitrary nodes of a Yutsis graph is linear in the number of nodes of the graph. Since the number of couples (i, j), with $i, j \in V$ is equal to $n(2n - 1) = O(n^2)$, the total number of paths in Y should be $O(n^3)$. Cycles constructed by the algorithm of Vismara are composed of two cyclic canonical paths of equal length from $r \to x$ (even cycle) or two cyclic canonical paths from $r \to p$, $r \to q$ and the edge (p, q) (odd cycle), with $r, p, q, x \in V$, $(p, q) \in E$ and $\min(r, p, q, x) = r$. When r and x are fixed, we can combine two paths out of a set of $O(n)$ paths from $r \to x$ to form an even cycle, bounding the number of even cycles determined by r and x to $O(n^2)$. This makes the total number of even cycles $O(n^4)$. Similarly the number of odd cycles for fixed r, p and q is also $O(n^2)$. Since the number of edges is also $O(n)$ the total number of odd cycles is also $O(n^4)$, making the total number of cycles generated by the algorithm of Vismara $O(n^4)$. In [17] some pathetic cases are constructed which have an exponential number of cyclic canonical paths. In Figure 3 such a cubic graph is shown, having $2^{\frac{n}{3}+1}$ cyclic canonical shortest paths between the nodes 1 and $2n$. This graph has in total $\frac{n}{18}(612^{\frac{n}{3}+1} - 86)$ shortest paths.

Since the assumption that the number of shortest paths between two arbitrary nodes is linear in the number of nodes for an average Yutsis graph is hard to

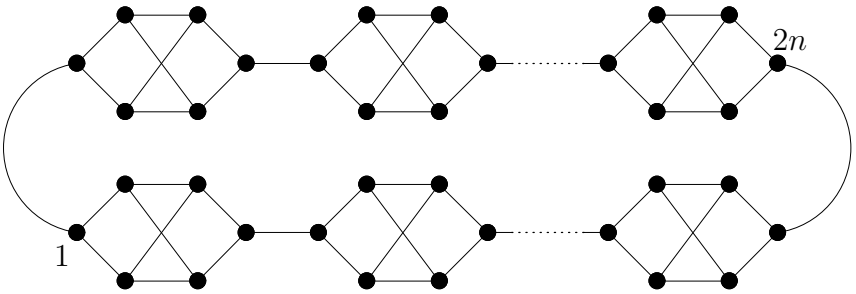


Fig. 3. A cubic graph having $2^{\frac{n}{3}+1}$ cyclic canonical paths between the nodes 1 and $2n$. All other nodes are labeled between 1 and $2n$. Note that this graph is not Yutsis.

prove, we calculated the number of paths and cycles for all Yutsis graphs upto $n = 11$. For more details on how we calculated these sets, we refer to [15]. Since these graphs are still quite small for experimental asymptotic analysis, we also generated some random test sets of large cubic graphs with McKay's program `genrang`, a part of the nauty package [14]. With this program one can generate an arbitrary number of random cubic graphs of given size, in which each labeled cubic graph has an equal probability to appear. This has the disadvantage that the graphs have a probability proportional to the size of their automorphism group, but for our purposes this should suffice. We generated each time $10n$ graphs of size n (corresponding with $2n$ nodes and $3n$ edges). The results are shown in Table 1.

The total number of paths of a cubic graph seems to be $\Omega(n^2)$ but $O(n^3)$. The total number of cyclic canonical paths on the other hand seems to be $O(n^2)$. To make this more clear, we divided the results by n^2. The number of cycles is clearly $O(n^4)$; the results even indicate a possible $O(n^2)$ behavior. All these results are on average.

Note that the random graphs in Table 1 are not necessarily Yutsis. In [15] we calculated the number of Yutsis graphs within the set of (bridgeless) cubic graphs upto $n = 13$. The ratio of the number of non-Yutsis graphs on the number of cubic graphs (including those having a bridge) decreases. For $n = 13$, corresponding with graphs with 26 nodes and 39 edges, only 1.23% of the cubic graphs are not Yutsis. It is thus reasonable to assume that for high values of n the probability that a random cubic graph is Yutsis tends to 1.

3.3 Algorithm complexity

With the assumptions of the foregoing paragraph in mind, generating the relevant cycles of a cubic graph can be done in $O(n^5)$ time. Hence, if we assume that the cost $h(n)$ of the heuristic is not more than $O(n^5)$ (which is the case for the heuristic in Section 4, but not for the heuristics in Section 5),

n	$E[P]/n^2$	$E[P_C]/n^2$	P_{max}	$P_{C,max}$	$E[C]/n^2$
4	3.85	2.63	6	4	0.788
5	3.94	3.10	4	4	0.616
6	4.23	3.32	8	6	0.587
7	4.38	3.42	8	6	0.544
8	4.53	3.50	16	12	0.509
9	4.64	3.54	20	16	0.480
10	4.74	3.55	24	20	0.456
11	4.83	3.55	32	24	0.434
10	4.68	2.71	10	7	0.329
50	5.52	1.97	19	15	0.184
100	5.64	1.70	25	20	0.157
150	5.69	1.57	22	21	0.145
200	5.71	1.49	26	20	0.138
250	5.72	1.43	26	20	0.133
300	5.73	1.38	26	20	0.129
350	5.74	1.35	26	22	0.126
400	5.74	1.32	27	19	0.123
450	5.74	1.29	28	24	0.121

Table 1
The average number of (cyclic canonical) paths ($E[P]$), the maximum number of paths between two arbitrary nodes (P_{max}) and the average number of cycles ($E[C]$) generated by the algorithm of Vismara for for all Yutsis graphs on 8 to 22 nodes (upper part) and for $10n$ random generated cubic graphs on $2n$ nodes (lower part). The subscript $_C$ stands for cyclic canonical.

the generation of the relevant cycles is the heaviest part of the algorithm, and one execution of one step of the while-loop takes $O(n^5)$ time. Otherwise $h(n)$ will dominate the execution of one step in the while-loop making it $O(h(n))$.

Estimating an upper bound for the number of reduction steps needed to reduce a Yutsis graph can be done by referring to another method for calculating general recoupling coefficients described in [5] and [6]. This method uses so-called rotations to transform the tree corresponding to the left-hand side of the general recoupling coefficient, called a binary coupling tree, into the tree corresponding to the right-hand side of the general recoupling coefficient. For each rotation used in this method of trees an equivalent interchange can be

constructed in the graphical method. In this way every solution found by the method of trees can be translated into a sequence of interchanges which form a solution in the graphical method. However the reverse is not true, making the graphical method more powerful than the method of trees.

In [6] the authors construct a rotation graph G_n defined as the graph of all binary coupling trees on $n+1$ leaves, with edges connecting trees that can be transformed into each other by a single rotation, and they prove that the diameter of this graph is bounded by:

$$\frac{1}{4} n \lg(n!) < \operatorname{diam}(G_n) \leq n\lceil \lg(n) \rceil - 2^{\lceil \lg(n) \rceil} + 1 + 2(n - \lceil \lg(n+1) \rceil) .$$

This means that the transformation of an arbitrary binary coupling tree on $n+1$ leaves to another binary coupling tree on $n+1$ leaves requires at most $O(n \log n)$ rotations. For a Yutsis graph of $2n$ nodes and $3n$ edges, this means that the number of interchanges in the most efficient reduction is at most $O(n \log n)$.

The proof in [6] constructs a sequence of $O(n \log n)$ rotations, consisting of $O(n)$ operations to transform each tree into a *spine* (a tree of maximum height) and $O(n \log n)$ operations to transform one spine into the other. Obviously this path (in the rotation graph) from one tree to the other is not necessarily the shortest path between the two trees, and hence does not correspond to the most efficient reduction. Note again that this path can be translated immediately in terms of interchanges on a Yutsis graph, corresponding to a reduction using $O(n \log n)$ steps. Hence, taking into account the fact that we can use heuristics to guide the reduction process in an intelligent way, it is not unreasonable to expect that the algorithm `CCA` combined with appropriate heuristics, will find a shorter reduction than the path described above, and thus will need only $O(n \log n)$ reduction steps, resulting in a total complexity of $O(\max(n^5, h(n))n \log n)$ for the algorithm.

4 Improving the heuristic using edge costs (`CCA-EC`)

This is the heuristic described in [16].

The idea here is to associate a cost with each cycle of smallest length in the graph and then to select a cycle with minimal cost. We define the cost of an edge as the difference in length of the two smallest cycles in which the edge participates. The cost of a cycle is then the minimum of the cost of its edges. Having computed the cost for all the girth cycles we select the girth cycle with minimal cost. When more than one girth cycle with minimal cost

exists, we select the cycle for which this minimum edge cost is most reached. If that still leaves more than one candidate, we select the cycle with the lowest sum of all its edge costs. If still more than one candidate remains, we select a cycle at random from these candidates. In the selected cycle we select an edge with minimal cost, again when there are several candidates one at random is chosen. On this edge we perform an interchange in such a way that both cycles in which the chosen edge participates become one unit smaller.

The time cost $h(n)$ of this heuristic is $O(n^5)$: to assign the edge costs to each edge, we have to process all the edges of each generated cycle. Since we generate $O(n^4)$ cycles with a $O(n)$ cost to process each cycle, the total cost of the heuristic is $O(n^5)$.

In order to check the performance of the heuristic `CCA-EC` we implemented a hybrid algorithm that allows to search the solution tree guided by a heuristic. For this purpose we used simplified versions of the reduction rules, i.e. without formula generation and dropping the node signs and edge directions, making the reduction process significantly lighter for this exhaustive search.

In principle this algorithm performs a depth-first search of the solution tree, taking into account all possible operations at each point, but giving preference to the operations the heuristic would choose. This is done by first collecting the operations the heuristic selects and placing them in front of a list. Afterwards all possible operations are added to the list, filtering out the operations the heuristic returned. An obvious upper bound on the length of the reduction, and hence on the depth of the solution tree, is given by $D - 1$, with D the number of interchanges used by `CCA-EC`. In addition we apply the branch-and-bound technique [1] to prune branches not containing shortest reductions. A cheap minimum cost function can be created using the following observations: (1) a Yutsis graph having a triangle can be reduced with one interchange if it is equal to a K_4 [2], otherwise at least 2 interchanges will be needed; (2) a Yutsis graph having no triangle needs at least 3 interchanges to reduce.

Using this approach we confirmed for all small cases (upto 10 nodes) that the reduction obtained by `CCA-EC` is indeed minimal w.r.t. the number of interchanges, i.e. it generates the best possible formula w.r.t. the number of summation variables and products of 6-j symbols. Furthermore the optimality of these small cases can be used incrementally in the search process: once the graph is small enough we just apply the heuristic, knowing it returns a minimal reduction.

For cases where exhaustive search is computationally unfeasible, we use a technique called *limited discrepancy search* (LDS), introduced in [11], to find

[2] K_n denotes the complete graph on n nodes. A complete graph is a graph in which any two nodes are connected.

shorter reductions. The key idea of LDS is that the heuristic would have found a good reduction if it had not taken a few "wrong turns". For a solution tree of height d, there are only d ways that the heuristic could make one wrong turn and $\binom{d}{k}$ ways it could make k. If k is small, a good reduction can be found by systematically searching all paths in the solution tree that differ from the heuristic path in at most a small number of decision points or *discrepancies*. LDS is a backtracking algorithm that searches the nodes of the solution tree in increasing order of such discrepancies.

Using LDS on the algorithm `CCA-EC` we obtained examples where `CCA-EC` does not return an optimal formula. Studying these examples gave rise to the new heuristics we describe in Section 5. Moreover, we also obtained a counter-example showing that the widespread convention of reducing a girth cycle in each step is not always the best thing to do. LDS found a reduction of Cg_7 using only 25 interchanges, where `CCA-EC` needed 26 interchanges. In the sequence found by LDS the third operation was an interchange making a 6-cycle bigger, while a 5-cycle was available. However, none of the possible interchanges involving this 5-cycle led to a graph that could then be reduced with 22 interchanges. Hence the interchange involving the 6-cyle was obviously a better thing to do at this point.

5 Heuristics with counting of cycles (`CCA-BS` and `CCA-CC`)

The heuristic `CCA-EC` is designed to reduce two cycles in length by performing only one interchange. Its key idea is that an edge of a cycle with minimal cost will not only reduce this cycle but also at least one neighboring cycle with which it shares the edge. However, this is not always true: in the case where both cycles share a path of length $l > 2$ instead of a single edge, it is impossible to perform an interchange reducing both cycles when one of the endpoints of the edge is one of the endpoints of the path. This situation is shown in Figure 4. In the special case where a path of length 2 is shared, every interchange on one of the edges reduces only one cycle. This situation is often found in cases with higher girth. A detailed study of this testcase (Cg_7) revealed that the problem mentioned in the previous section is caused by the same effect.

An improvement of the original heuristic is based on the following observation. In cases with higher girth it is often possible to reduce more than 2 small cycles in length by one interchange, if an edge is shared by more than 2 small cycles. E.g. on Cg_7 we can perform an interchange reducing 4 girth cycles at once. Such a situation can be detected by looking at the effect of all interchanges reducing a girth cycle in length. It is easy to see that an interchange on an edge e interchanging b and c, as shown in Rule III of Figure 1, will reduce a

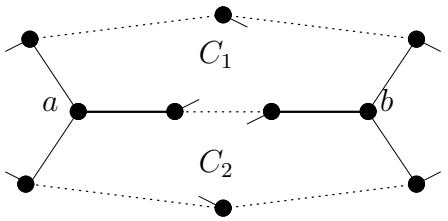


Fig. 4. Two cycles C_1, C_2 sharing a path from a to b. It is impossible to perform an interchange on the bold edges reducing both cycles in length.

cycle in length if and only if e is in the cycle together with one of $\{b, c\}$. The same operation will make the cycle bigger if and only if e is not in the cycle, but b or c is.

We can calculate this effect on all relevant cycles by walking over all the girth cycles, collecting every interchange making a girth cycle smaller exactly once and calculating its effect by walking over all relevant cycles. Let S_l, resp. B_l, be the set of relevant cycles of length l which become smaller, resp. bigger, by performing some interchange. Then we can compare two interchanges I and I' by comparing the sets S_l and S'_l, resp. B_l and B'_l, using one of the following criteria:

More Smaller/Less Bigger (`CCA-SB`): We prefer I if $S_l > S'_l$ or, when $S_l = S'_l$, if $B_l < B'_l$.

Cycle Count (`CCA-CC`): Let C_l be the number of cycles of length l and C_l^I the number of cycles of length l after interchange I is performed, then $C_l^I = C_l - S_l - B_l + S_{l+1} + B_{l-1}$. We prefer I if $C_l^I > C_l^{I'}$.

Starting with l equal to the girth, we repeat the chosen criterium until a difference is found or until l becomes bigger than the biggest relevant cycle, in which case both interchanges are considered equivalent.

The time cost $h(n)$ of both heuristics is $O(n^{10})$: to construct the sets S_l and B_l for a given interchange I and each applicable l we need to look at each edge of each generated cycle, resulting in $O(n^5)$ edges to be processed. For each edge of a girth cycle, we collect an interchange making the girth cycle smaller, yielding $O(n^5)$ operations.

6 Program description

Since the program is a new version of the `GYutsis` program described in [16], we will focus on what is changed in comparison with the previous version.

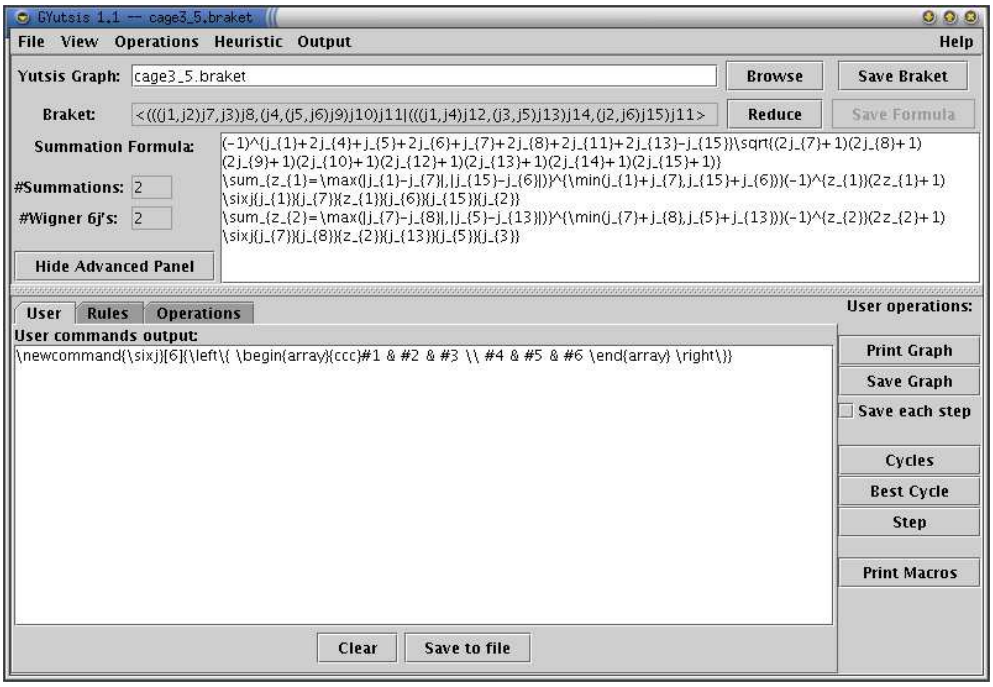


Fig. 5. The GYutsis program in advanced mode after performing two steps in the reduction of Cg_5, with LaTeX as outputformat. The macro for the Wigner 6-j symbol was printed to the "*User*" pane by pressing the "*Print Macros*" button.

6.1 New features and use of the program

In order to use the program, Sun's Java Runtime Environment version 1.4 or higher, must be installed on your system. To use the graphical front end `GYutsis`, a graphical environment such as the X-window system (Linux/Unix and friends), MacOs or MS Windows is needed.

The program is started with the command `java -jar GYutsis.jar [<arg>]`, where the optional argument is a braket or the name of a file containing a Yutsis Graph in the `BRAKET` or `YTS` format. For details on the file formats we refer to the documentation provided with the program. The documentation can be viewed from the program itself by selecting "*Help*" from the "*Help*" menu or by viewing the file `GYutsisHelp.html` with your favourite browser. The `readme` file explains how the file can be extracted from the jar file. Figure 5 shows a screenshot of the program.

Like the previous version, the program can be used in two modes: normal and advanced mode. In advanced mode an additional panel is attached to the main window, to which we refer as the *advanced panel*. One can switch from normal to advanced mode and vice versa by pressing the "*Show/Hide Advanded Panel*" button or by selecting "*Show/Hide Advanded Panel*" from the "*View*" menu.

A problem is defined by entering a braket or filename in the textfield at the top of the window. Once a problem is defined, the initial formula (see equation (2)) is immediately generated (and shown in advanced mode, but not in normal mode) and the "*Reduce*" button becomes active. From the "*Heuristic*" menu one can select the heuristic to be used by the algorithm and from the "*Output*" menu the desired output format of the formula. Four formats are available:

- `Generic`: a default human readable format (this is the format used by the previous version).
- `LaTeX`: LaTeX format. One can choose here to use a macro to represent the Wigner 6-j symbol or generate it in plain LaTeX.
- `Maple`: generic format for the popular computer algebra package Maple (see [12]), macros containing Maple functions for the Kronecker Delta, triangular and Wigner 6-j symbol are always used.
- `Racah`: format compatible with the Maple package `RACAH`, which is a package especially for Racah algebra (see [8]).

The macros used for the LaTeX and Maple format can be generated to the "*User*" pane in advanced mode by pressing the "*Print Macros*" button or by selecting the corresponding item in the "*Output*" menu. By pressing the "*Save to file*" button on the "*User*" pane, one can optionally save it to a file.

By pressing the "*Reduce*" button the formula is generated and shown in the selected format.

In this version it is also possible to save the file graphically on demand, by selecting the "*Save Graph*" button, or after each performed step, by checking the "*Save Each Step*" checkbox. Both actions can also be selected from the "*Operations*" menu. The graph is then saved in gml-format for the graph drawing tool `Graphlet` (see [10]), which is freely available for academic and educational use.

There is also an applet version, `GYutsisApplet`, making it possible to run the program from your favorite webbrowser without installing it on your computer. The functionality is similar, but without all file operations. It can be run from the URL `http://caagt.rug.ac.be/yutsis/GYutsisApplet.caagt` (normal mode) or `http://caagt.rug.ac.be/yutsis/GYutsisAdvanced.caagt` (advanced mode). In order to be able to run the applet, your browser needs the Java plugin from Sun, version 1.4 or higher.

In the applet version of the program we also included some example problems, which can be selected from the example menu, which are interesting to compare the different heuristics.

The program is written in Java 1.2 and consists in total of 33 classes. The classes used to generate the paths and cycles and to represent the generated formula only had some minor changes, so we refer to [16] for their description. We will focus here on the structural changes of the application.

We created a new abstract class `AbstractGraph` bundeling common functionality for all classes implementing the `Graph` interface, minimizing the effort to write a class implementing this interface by subclassing from this class. In a similar way we also introduced a class `AbstractYutsis`, a subclass from `AbstractGraph`, providing some functionality of the `Yutsis` interface. The aim is to use this class as a base class for classes implementing the `Yutsis` interface. `YutsisGraph` is now a subclass of `AbstractYutsis`.

The heuristics follow the interface `CCAHeuristic` and are all subclasses of `AbstractCCAHeuristic`. The `EC` heuristic is implemented in the `EdgeCostHeuristic` class, the heuristics `SB` and `CC` in the `CycleCountHeuristic` class. By default this last class uses the `SB` heuristic, but this can be changed by calling the `setStrategy(int)` method with `CycleCountHeuristic.CYCLE_COUNT` as argument.

A class diagram [7] highlighting the design changes of the application is shown in Figure 6.

Different representations of the `GenRecoupCoeff` class are provided by means of the well known *Visitor pattern* [9], implemented by the `GRVisitor` interface. Again we created an abstract class `AbstractGRWrappedTextVisitor` providing basic wrapping functionality. The four delivered visitors are all subclasses of this class:

- `GRWrappedStringVisitor` delivers a default String representation. This is the representation used in the previous version.
- `GRWrappedLaTeXVisitor` delivers a LaTeX representation of the generated formula. One can choose to use a macro for the Wigner 6-j symbol.
- `GRWrappedMapleVisitor` delivers a plain Maple representation of the summation formula. In this representation macros are always used to represent the Kronecker delta, triangular and Wigner 6-j symbol. These macros are in fact Maple functions.
- `GRWrappedRacahVisitor` delivers a representation compatible with the Maple package `RACAH`, a package especially for Racah algebra ([8]).

A default non wrapped representation is still available by the `GenRecoupCoeff`'s `toString()` method, which is called automatically when the object is printed to a `PrintStream`.

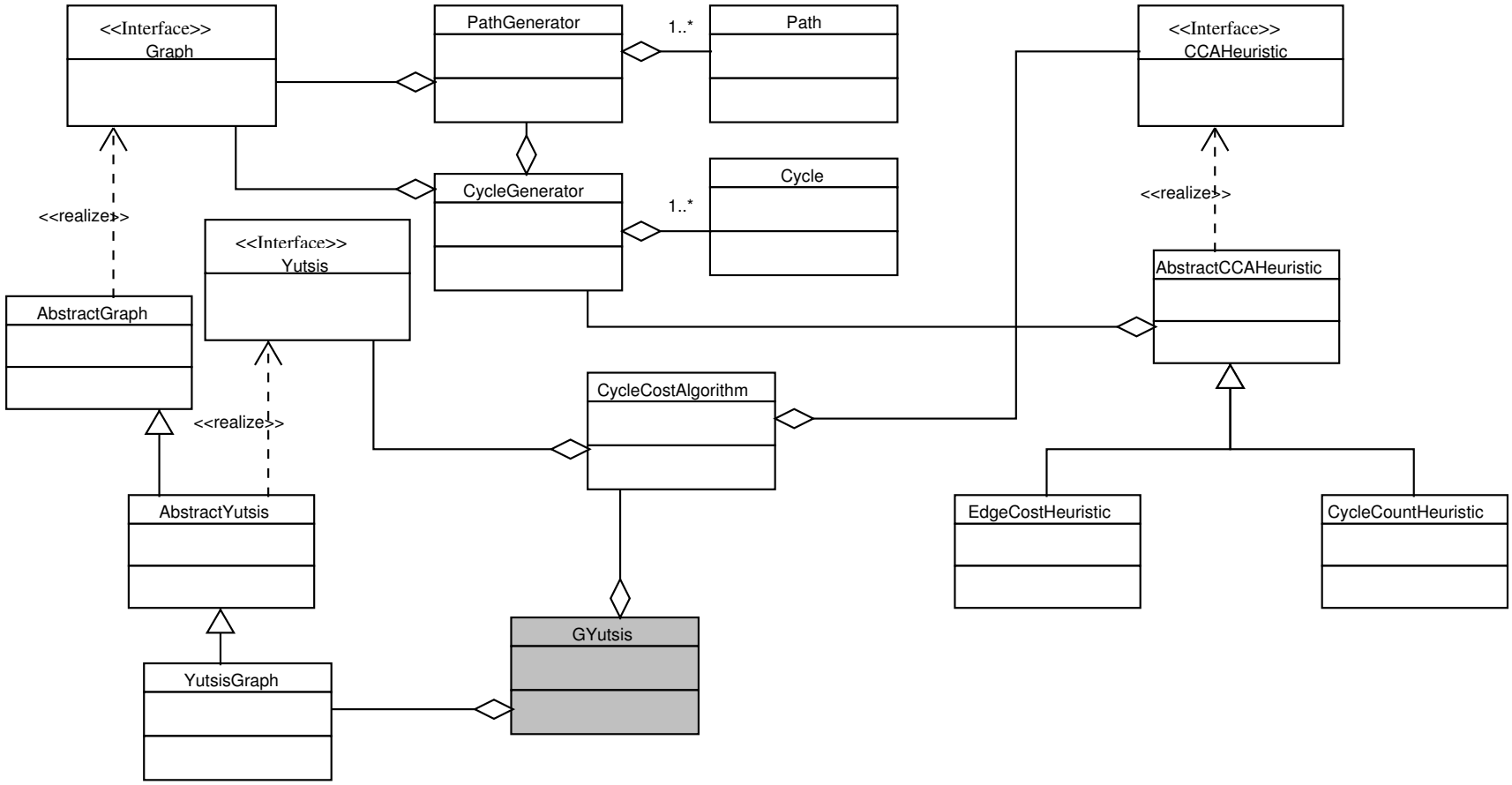


Fig. 6. Class diagram of the `CycleCostAlgorithm` application and its graphical front end `GYutsis` highlighting the structural changes.

If one wants to use `CycleCostAlgorithm` to calculate a given general recoupling coefficient, the following code suffices to obtain the formula as a `GenRecoupCoeff` object:

```
YutsisGraph y = new YutsisGraph("<BRA|KET>");
CCAHeuristic h;
if (choice == EC)
   h = new EdgeCostHeuristic(y);
else if (choice == SB) // More Smaller/Less Bigger
   h = new CycleCountHeuristic(y);
else{ // CC
   CycleCountHeuristic cch = new CycleCountHeuristic(y);
   cch.setStrategy(CycleCountHeuristic.CYCLE_COUNT);
   h = cch;
}
CycleCostAlgorithm cca = new CycleCostAlgorithm(y,h);
cca.reduce();
GenRecoupCoeff grc = y.genRecoupCoeff();
```

In order to do something with this `GenRecoupCoeff` one should write a class implementing the `GRVisitor` interface retrieving the necessary information. For each type of object in a `GenRecoupCoeff` there is a method `visit<type>(<type>)` in the `GRVisitor` interface. Each object that accepts the visitor by means of the `accept(GRVisitor)` method calls the visit method for his type. Compounded objects let the visitor visit first themselves and afterwards there composites. This is illustrated in the sequence diagram [7] in Figure 7. In Figure 8 a class diagram of the classes representing the summation formula as a `GenRecoupCoeff` object is shown, together with the methods one needs to retrieve the information concerning the formula from these classes. For a full description on these methods we refer to the documentation delivered with the program source.

Writing an implementation of `GRVisitor` is fairly easy: one has to fill in the code to process each type by means of the `visit<type>` method, knowing the sequence in which each object in a `GenRecoupCoeff` is visited and return the result by the `result()` method. In each `visit<type>` method a reference to the object is passed, allowing the `GRVisitor` to collect the needed information to process the object.

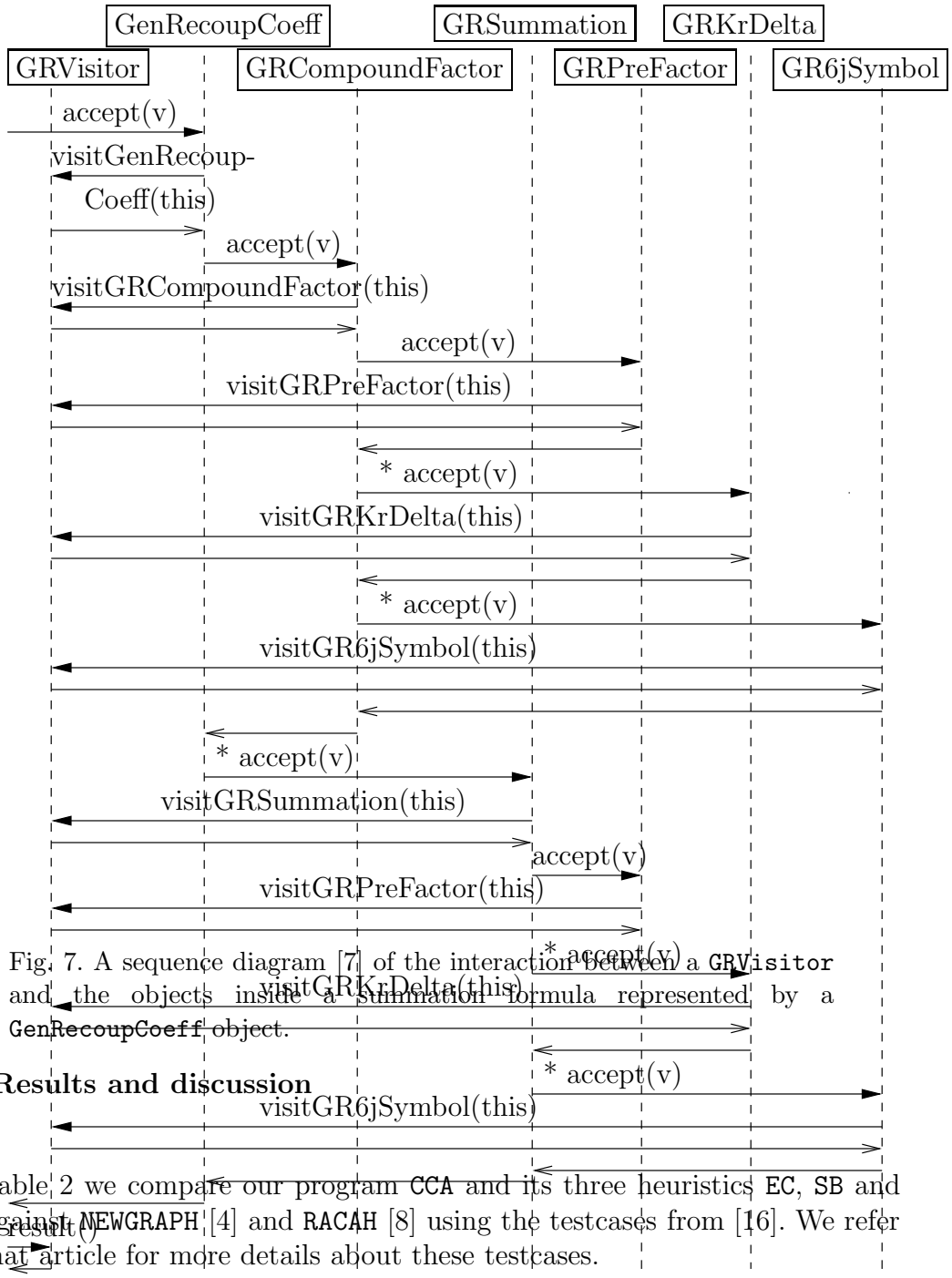


Fig. 7. A sequence diagram [7] of the interaction between a `GRVisitor` and the objects inside a summation formula represented by a `GenRecoupCoeff` object.

7 Results and discussion

In Table 2 we compare our program `CCA` and its three heuristics `EC`, `SB` and `CC` against `NEWGRAPH` [4] and `RACAH` [8] using the testcases from [16]. We refer to that article for more details about these testcases.

Clearly, `CCA` outperforms `NEWGRAPH` and `RACAH` with all three heuristics, but in order to compare the heuristics with each other we need harder cases: only on the cases Cg_8 [3], Cb_6 and Cb_8 the new heuristics `SB` and `CC` do better than `EC`. A difference between `CCA-SB` and `CCA-CC` can not be found on these "small" cases. It is interesting to note however, that both heuristic algorithms obtain

[3] In [16] these cases were labeled C_g, but since this is the standard notation for a cycle of length g we prefer to more descriptive label Cg_g.

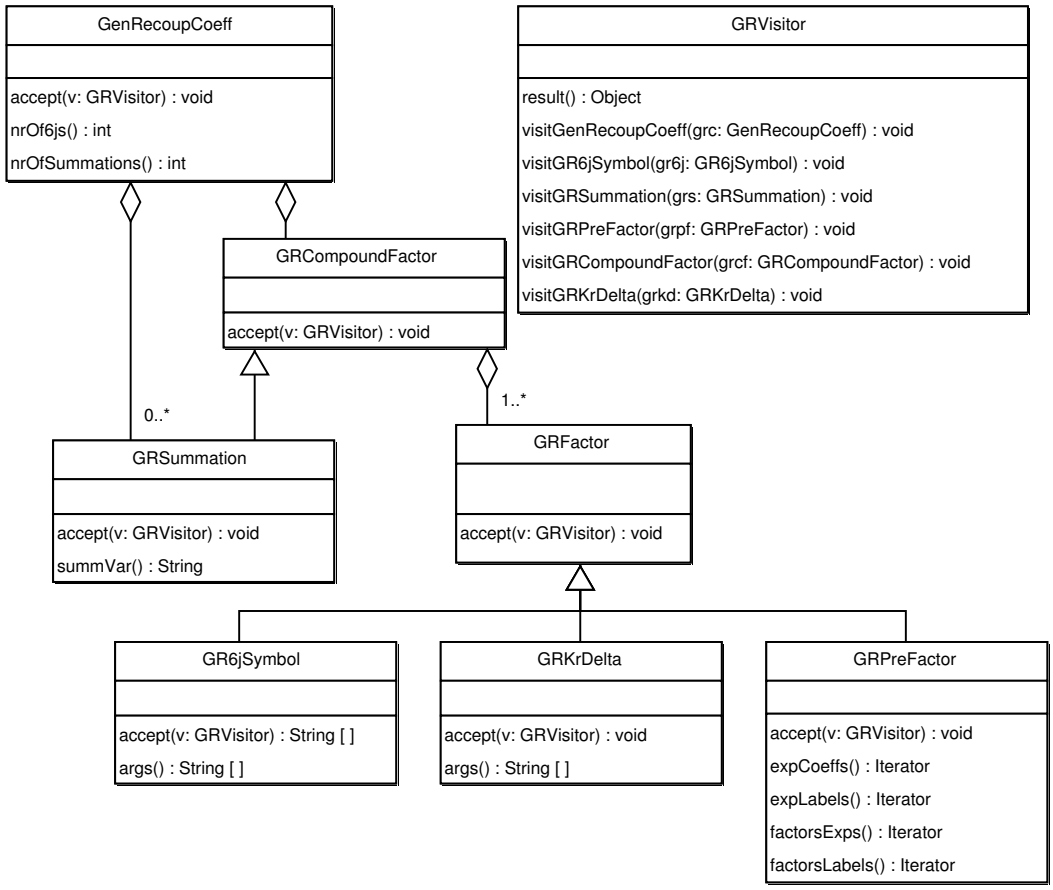


Fig. 8. A class diagram [7] of the classes used to represent a summation formula for a general recoupling coefficient. The shown methods can be used to retrieve all the information about the generated summation formula.

the same result as `CCA-EC` for the case Cg_7, the case where LDS found a better reduction.

Since the girth is a good indicator for the complexity of the underlying recoupling coefficient, we choose all 18 cubic cages[4] of girth 9, labeled $Cg_{9,i}$, $i = 1, \ldots, 18$ and having 58 nodes, as testcases for comparing the heuristics with each other. The results are shown in Table 3.

From these experiments it is clear that for small cases (say upto 24 nodes or 36 j's) the results obtained by `CCA-EC` are as good as those obtained by `CCA-SB` and `CCA-CC`, while for bigger cases `CCA-SB` or `CCA-CC` often returns better results than `CCA-EC`. Note that the heuristic `CCA-EC` is cheaper ($h(n) = O(n^5)$) than the other heuristics ($h(n) = O(n^{10})$). Hence, the overall time complexity of `CCA-SB` and `CCA-CC` is $O(n^{11} \log n)$. The experiments give no clear indication which of the heuristics `CCA-SB` or `CCA-CC` is best. We do know however that

[4] The cubic cage of girth g is the smallest cubic graph having girth g.

Test case	#j's	girth	NEWGRAPH	RACAH	CCA-EC	CCA-SB	CCA-CC
(G_1)	9	3	2	2	2	2	2
(G_2)	18	3	2	2	2	2	2
(G_4)	18	4	10	13	10	10	10
(F_0)	9	4	3	3	3	3	3
(F_1)	12	4	4	5	4	4	4
(F_2)	15	3	5	6	5	5	5
(F_3)	15	4	6	6	6	6	6
(F_4)	15	4	6	6	6	6	6
(F_5)	18	4	7	10	7	7	7
(F_6)	18	3	7	8	7	7	7
(F_7)	18	4	9	10	8	8	8
(F_8)	21	4	12	12	10	10	10
(F_9)	27	5	16	18	15	15	15
(F_{10})	33	4	12	12	12	12	12
(F_{11})	36	4	15	16	14	14	14
(F_{12})	42	4	21	24	18	18	18
(Cg_5)	15	5	7	7	7	7	7
(Cg_6)	21	6	–	–	12	12	12
(Cg_7)	36	7	–	–	26	26	26
(Cg_8)	45	8	–	–	38	37	37
(Cb_6)	24	6	–	17	15	14	14
(Cb_8)	51	8	–	–	44	42	42

Table 2
Number of interchanges performed by NEWGRAPH, RACAH, CCA-EC, CCA-SB and CCA-CC.

none of the heuristics is optimal, since we found shorter reductions using LDS.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *"Data Structures and Algorithms"* (Addison-Wesley, 1983).

[2] A. Bar-Shalom and M. Klapisch, *Comput. Phys. Commun.* **50** (1988) 375.

Case	$(Cg_{9,1})$	$(Cg_{9,2})$	$(Cg_{9,3})$	$(Cg_{9,4})$	$(Cg_{9,5})$	$(Cg_{9,6})$
`CCA-EC`	87	89	89	88	89	92
`CCA-SB`	86	83	85	85	87	84
`CCA-CC`	86	86	84	87	86	85

Case	$(Cg_{9,7})$	$(Cg_{9,8})$	$(Cg_{9,9})$	$(Cg_{9,10})$	$(Cg_{9,11})$	$(Cg_{9,12})$
`CCA-EC`	90	90	91	93	90	91
`CCA-SB`	84	86	84	85	85	83
`CCA-CC`	84	85	83	85	85	86

Case	$(Cg_{9,13})$	$(Cg_{9,14})$	$(Cg_{9,15})$	$(Cg_{9,16})$	$(Cg_{9,17})$	$(Cg_{9,18})$
`CCA-EC`	88	89	91	91	94	91
`CCA-SB`	84	86	84	85	87	86
`CCA-CC`	85	86	85	85	87	86

Table 3
Number of interchanges performed by `CCA-EC`, `CCA-BS` and `CCA-CC` on all 18 cubic cages of girth 9.

[3] L.C. Biedenharn and J.D. Louck, "Coupling of n angular momenta: recoupling theory", in: *The Racah-Wigner Algebra in Quantum Theory, Encyclopedia of Mathematics and its Applications*, Vol. 9, pp. 435–481 (Addison-Wesley, 1981).

[4] V. Fack, S. N. Pitre and J. Van der Jeugt, *Comput. Phys. Commun.* **101** (1997) 155.

[5] V. Fack, S. Lievens and J. Van der Jeugt, *Comput. Phys. Commun.* **119** (1999) 99–114.

[6] V. Fack, S. Lievens and J. Van der Jeugt, *Discrete Mathematics* **245** (2002) 1–18.

[7] M. Fowler with K. Scott, "UML Distilled", (Addison Wesley, 1997),

[8] S. Fritzsche, T. Inghoff, T. Bastug and M. Tomaselli, *Comput. Phys. Commun.* **139** (2001) 314.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns", pp. 331-344 (Addison-Wesley, 1995).

[10] http://www.infosun.fmi.uni-passau.de/Graphlet/, Graphlet Homepage.

[11] W. D. Harvey, "Nonsystematic backtracking search", Phd Thesis, pp. 48–73 (Stanford University, 1995).

[12] A. Heck, "Introduction to Maple" Second Edition, (Springer-Verlag, 1996).

[13] P. M. Lima, *Comput. Phys. Commun.* **66** (1991) 89.

[14] B. McKay, Nauty homepage: http://cs.anu.edu.au/ bdm/nauty/.

[15] D. Van Dyck and V. Fack, "An efficient algorithm for filtering Yutsis graphs", submitted to *Journal of Graph Algorithms and Applications* (2002).

[16] D. Van Dyck and V. Fack, *Computer Physics Communications* **151** (2003) 353–368.

[17] P. Vismara, *The Electronic Journal of Combinatorics* Vol. 4(1) (1997) #R9.

[18] A.P. Yutsis, I.B. Levinson and V.V. Vanagas, *Mathematical Apparatus of the Theory of Angular Momentum* (Israel Program for Scientific Translation, Jerusalem, 1962).