On the reduction of Yutsis graphs

D. Van Dyck, V. Fack

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

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 gives an overview of the state-of-the-art heuristic algorithms, used in the latest version of our `GYutsis` program, for calculating general recoupling coefficients. By means of an experimental setup we show that, in particular for problems of higher order, this approach yields summation formulae which are at least as good, but are often more concise than those obtained by previous algorithms. We also give a counter-example showing that that the widespread convention of reducing girth cycles first does not always lead to a shortest reduction.

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

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 [1]. 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 [2–5]. 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 [6] 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 a factor of the final summation formula.

The first algorithms implementing these graphical techniques were developed by Bar-Shalom and Klapisch [2] and later by Lima [3] who added a limited possibility to manipulate the resulted expressions. Their main technique was to eliminate smaller cycles first from the graph, since smaller cycles give a smaller contribution to the formula. Fack *et al.* [4] also adopted this idea in their algorithm `NEWGRAPH`, which uses a heuristic to select a 4-cycle out of several available 4-cycles for reduction, and thus results in more concise expressions. However, a counter-example presented in this paper will show that reducing the smallest cycle first does not always lead to the most concise summation formula.

More recent work on the subject was done by Fritzsche *et al.* [5] who implemented the graphical techniques in their Maple-package `RACAH`, allowing the expressions to be transformed algebraically and evaluated in a broader Racah algebra package, but not leading to better expressions.

As pointed out in [4] the quality of the generated formula depends strongly on which rule is selected in each step and on which cycle of the graph it is applied. In this article we give an overview of the heuristics, used in our `CycleCostAlgorithm` (`CCA`) algorithm and `GYutsis` program, for calculating general recoupling coefficients. All these heuristics examine the cyclic structure of the graph in order to decide which rule to apply on which cycle. As will be clear from our experiments, these approaches yield significantly better formulae, in particular for problems of higher order.

We analyze the weak points of the `EdgeCostHeuristic` (`EC`) heuristic described in [7] and introduce two new heuristics which remedy these problems. For a more thorough treatment of the assumptions made to bound the algorithm complexity, the implementation details, features and the use (both for end users and application programmers) of the `GYutsis` program we refer to [8]. The program itself can be downloaded from our website `http://caagt.ugent.be/yutsis/` under the section "Software". On the same page a limited applet version of the program is available which can be run by a browser which has Sun's Java plugin 1.4 (or higher) installed.

The paper is organized as follows. Section 2 recalls some notions from the theory of angular momentum theory and Yutsis graphs. Section 3 describes the ideas behind the generic algorithm `CCA` and discusses some properties of the algorithm. In Section 4 the heuristic `EC` based on edge costs, first introduced in [7], is described and its performance is investigated. Two more powerful heuristics are described in Section 5. Section 6 gives some conclusions.

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 [1] and [6].

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. As an example Figure 1 shows the resulting Yutsis graph with 6 nodes and 9 edges for the recoupling coefficient $F_0 = \langle\, ((j_1, j_2)j_5, (j_3, j_4)j_6)j_7) \mid ((j_1, j_3)j_8, (j_2, j_4)j_9)j_7 \,\rangle$.

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 di-

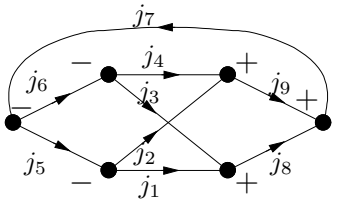


Fig. 1. Yutsis graph corresponding with the recoupling coefficient F_0.

agram multiplied by (see [6], 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.

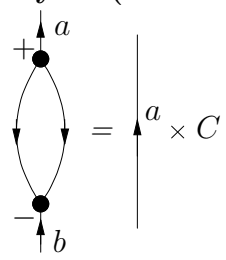
Once the graph is generated, it can be simplified with the help of the reduction rules developed by Yutsis, Levinson and Vanagas [6]. Figure 2 shows a graphical interpretation of these rules: reduction of a bubble, reduction of a triangle, interchange operation and reduction of a square. An interchange (rule III) on an edge e introduces a new label which replaces e (f in the case of rule III) which has to be summed over all possible values it can assume.

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. Moreover the reduction of a square can be seen as an interchange followed by the reduction of a triangle or, equivalently, as a sequence of two interchanges followed by the reduction of a bubble. This is why a summation over the new label f also appears in rule IV. There is no summation in rule II since the new label is removed by the $\delta(a,b)$ fixing the new label to the value a (b is the summation variable in the decomposition of a triangle).

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, as shown in Figure 3.

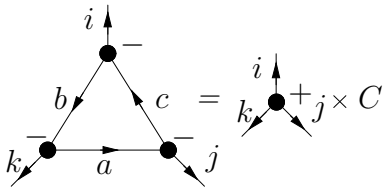
From an algorithmic point of view we are only interested in the complexity of the formula. Since the difference between the number of summations and the number of 6-j coefficients is constant, the number of 6-j coefficients suffices as measure for the complexity of the formula. With this idea in mind 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

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



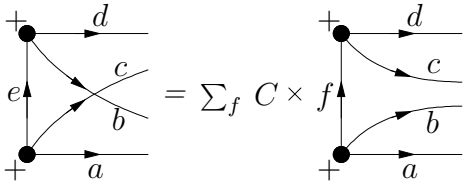
with $C = (2a+1)^{-1}\delta(a,b)$.

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



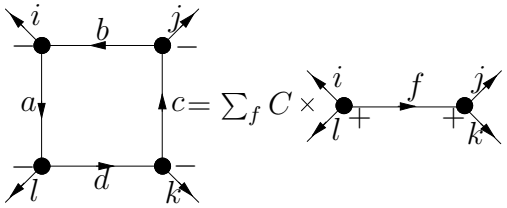
with $C = \left\{ \begin{array}{ccc} i & j & k \\ a & b & c \end{array} \right\}$.

Rule III: interchange



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

Rule IV: reduction of a 4-cycle (square)



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

Fig. 2. Graphical reduction rules for Yutsis graphs.

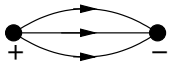


Fig. 3. Graphical representation of a triangular delta.

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.

Note also that every reduction of a Yutsis graph ends with an interchange on an edge of K_4, the complete graph on 4 nodes, assuming no bubble is present in the graph. Indeed, the only operation that can create a bubble is the reduction of a triangle in a *diamond*, a subgraph consisting of two triangles sharing an edge. Since the reduction of a triangle removes two nodes and cannot create more than one bubble, at most 4 nodes disappear per interchange. So the last graph on which an interchange is performed bigger than K_4 must have 6 or 8 nodes and no bubbles. Only the graph composed of two connected diamonds of those graphs contains a diamond. Reducing a triangle on this graphs creates K_4. Any other graph does not contain a diamond and thus cannot contain a bubble after performing an interchange. Since K_4 is the unique bubbleless cubic graph on 4 nodes, the last interchange must be performed on K_4. In Section 4 we will use this property to construct a cheap pruning function in an exhaustive search.

3 Algorithm for reducing Yutsis graphs (`CCA`)

This section describes the generic algorithm `CycleCostAlgorithm` (`CCA`), which eliminates cycles from the Yutsis graph step by step, until a triangular delta is obtained. Each reduction step reduces a smallest cycle in length. When several choices are possible, heuristics are used to select a 'best choice'. Details for the used heuristics are given in Sections 4 and 5.

We recall from graph theory that the size of the smallest cycle of a graph is called its *girth*. A *girth cycle* is a cycle whose length equals the girth of the graph.

In order to reduce a Yutsis graph we need at least its girth cycles. However, in order to make an 'intelligent' choice, the heuristics need to examine more of the cyclic structure of the graph. Determining cycles has been the bottleneck for all the older algorithms: for cases with girth greater than 4, most of the running time of the algorithm is spent in looking for cycles. They all work along the same principle, i.e. using the back edges of a *palm tree*.

A palm tree P of a graph G is a directed graph, consisting of two disjoint edge sets: the *front* edges and the *back* edges [1] [9]. The front edges form a

[1] Tarjan calls the back edges *fronds* in his article, but nowadays the term *back* edges is generally accepted and in our opinion more descriptive.

rooted spanning tree T of P, created by a depth-first search of the graph G and the edges have the direction they were traversed during the search. The back edges are the edges connecting the nodes with one of its ancestors in T.

Using the back edges of a palm tree, the algorithms first search for cycles of length 3, then for cycles of length 4, and so on, each step taking more time [2]. For small cycles this method is reasonably efficient, but for larger cycles a lot of back edges can be involved, making the number of different situations in which a cycle can be formed numerous and complicated.

Not only is this approach inefficient, it is also incorrect: some cycles will never be found [2], others are considered but are not relevant, since they can be decomposed into a sum of smaller cycles. The *sum* of two cycles C and C' is defined to be the symmetric set difference $(C \bigcup C') - (C \bigcap C')$, in which we consider a cycle as a set of edges [10] and the difference $A - B$ between two sets A and B as the set containing all the elements of A which are not in B. E.g. the sum of two squares sharing an edge will be a hexagon, which is clearly not relevant for our problem and should not be considered.

In [10] Vismara introduces the notion of cycle relevance: a cycle is called *relevant* if it belongs to at least one minimum cycle basis. In the case of unweighted graphs this means that a relevant cycle can not be decomposed into a sum of smaller cycles and vice versa. In the same article an algorithm is constructed to generate all the relevant cycle prototypes of a graph in polynomial time, which we refer to as the algorithm of Vismara. From these prototypes all relevant cycles can be easily generated. In fact it generates all cycles which can be decomposed in two shortest paths of equal length (an *even* cycle) or two shortest paths of equal length and an edge (an *odd* cycle). Even though every relevant cycle can be decomposed in this way, not all cycles having this property are relevant. The set of relevant cycles always contains all the girth cycles.

By using the algorithm of Vismara we can generate all the girth cycles, as well as provide the heuristics with good information on the cyclic structure on the graph, which allows them to decide which operation will be best without considering *all* cycles; moreover this information can be obtained in very acceptable time.

However, we do not always need an heuristic to select an operation: reducing a triangle only makes relevant cycles smaller and is as such always a good operation. 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) [10], 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. For this reason we can remove any triangle immediately without considering the effect of the operation on other cycles. The same reasoning can be made about bubbles.

As for the rule for reducing a square, we have chosen not to implement it and not to develop heuristics for its use. Keeping in mind the fact that the reduction for a square is nothing but an interchange followed by a reduction of a triangle, we let the heuristic decide how the square is transformed into a triangle. Implementing the square rule would make the algorithm heavier without gaining effiency since we need the relevant cycles of the graph anyway to make a good choice how to reduce the square. As such it is more convenient and efficient to treat a square as a general polygon. Note that implementing higher sum rules, like a pentagon and hexagon rule, would certainly not lead to better expressions, as pointed out in [7]. It would force an heuristic to perform a sequence of fixed interchanges followed by a triangle rule instead of letting the heuristic have the freedom of picking one interchange at a time, which makes it more powerful.

Summing up, Figure 4 gives a pseudocode formulation of the algorithm `CCA`.

```
CCA (YutsisGraph Y)
 1: while Y != triangular delta do
 2:    if Y has bubble then
 3:       Format and remove arbitrary bubble
 4:    else if Y has triangle then
 5:       Format and remove arbitrary triangle
 6:    else
 7:       Generate all relevant cycles (Vismara's algorithm)
 8:       Use heuristic to select the best cycle and its best edge
 9:       Format and interchange best edge out of the best cycle
10:    end if
11: end while
12: return formula
```

Fig. 4. Generic algorithm to generate a summation formula for a general recoupling coefficient from the corresponding Yutsis graph.

Next we consider the complexity of the algorithm. All relevant cycles of a cubic graph of order n can be generated in $O(n^5)$ time. This upper bound assumes that there are no more than n cyclic canonical shortest paths between two arbitrary nodes of the cubic graph. In [10] some pathetic cases are constructed which have an exponential number of cyclic canonical paths. In practice however, the bound of n cyclic canonical paths is seldom reached. For a more thorough treatment of this assumption we refer to [8].

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), the generation of the relevant cycles is the most expensive part of the algorithm, and one execution of the while-loop takes $O(n^5)$ time. Otherwise $h(n)$ will dominate the execution of 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 [11] and [12]. 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 [12] a rotation graph G_n is constructed 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 it is proven that the diameter of G_n is $\Theta(n \log n)$. Hence 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 [12] constructs a sequence of $O(n \log n)$ rotations which is not necessarily the shortest path (in G_n) between the two trees, and hence does not correspond to the most efficient reduction of the Yutsis graph. 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 an appropriate heuristic, 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 Heuristic using edge costs (`EC`)

The main idea behind this heuristic 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 costs of its edges. The key idea of this heuristic is that an interchange on

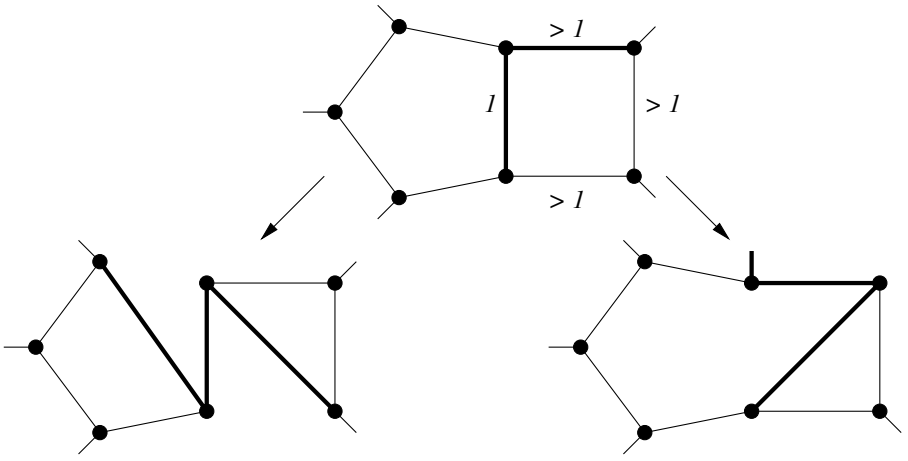


Fig. 5. A situation where NEWGRAPH could make a bad choice for reducing a square: in the middle the initial situation, on the left the choice CCA-EC would make on the basis of edge costs, on the right a possible choice of NEWGRAPH.

an edge making a cycle with minimal cost smaller, would not only reduce this cycle, but also at least one neighboring cycle with which it shares an edge.

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 the heuristic is $O(n^5)$.

We illustrate the behavior of this heuristic, especially as compared with the algorithm NEWGRAPH [4], with an example. Figure 5 shows a situation where the smallest cycle is a square sharing an edge with a pentagon, which is the second smallest cycle in the neighborhood. Since there are no neighboring squares, NEWGRAPH would arbitrarily choose between the two possible ways to reduce the square, both of which are shown in Figure 5. Obviously the CCA algorithm combined with the EC heuristic (short CCA-EC) would perform an interchange on the edge with lowest cost from the square (choice on the left in Figure 5) making the square *and* the pentagon one unit smaller. However a possible choice by NEWGRAPH could lead to the situation on the right in Figure 5, transforming the pentagon into a hexagon. (Note that in the drawing we did not reduce the triangle, which NEWGRAPH would do by applying the square rule, shown in Figure 2.) So, NEWGRAPH could yield one more summation variable and an extra Wigner 6-j symbol in the generated summation formula.

Case	(F_2)	(F_5)	(F_6)	(F_7)	(F_8)	(F_9)	(F_{11})	(F_{12})
#nodes	10	12	12	12	14	18	24	28
girth	3	4	3	4	4	5	4	4
`NEWGRAPH`	5	7	7	9	12	16	15	21
`RACAH`	6	10	8	10	12	18	16	24
`CCA-EC`	5	7	7	8	10	15	14	18

Case	(Cg_5)	(Cg_6)	(Cg_7)	(Cg_8)	(Cb_6)	(Cb_8)
#nodes	10	14	24	30	16	34
girth	5	6	7	8	6	8
`NEWGRAPH`	7	–	–	–	–	–
`RACAH`	7	–	–	–	–	–
`CCA-EC`	7	12	26	38	15	44

Table 1
Number of interchanges performed by `NEWGRAPH`, `RACAH` and `CCA-EC`.

In order to compare the results from the different algorithms, the list of testcases introduced in [13] is generally used. We refer to [7] for the definition of the testcases F_i $(i = 0, \ldots, 12)$. However, since the algorithms for calculating general recoupling coefficients become stronger and stronger, we added some more testcases of higher complexity and dropped the "easy" cases where all algorithms found equivalent reductions.

The cases Cg_g are the cubic cages of girth g, while the cases Cb_g are the minimal cubic non-cages of girth g. A *cubic cage* of girth g is a smallest cubic graph having girth g. Both classes are generated with Gunnar Brinkmann's program `MINIBAUM` [14]. We choose these graphs as testcases because the girth of the graph gives an idea of the complexity of the underlying recoupling coefficient. In order to map a recoupling coefficient on these graphs, we used an algorithm to find the partitioning of the nodeset decomposing the graph into two trees of equals size (see [15] for details). We choose an arbitrary edge on the cut and label it as the total angular momentum, label all the other edges on the cut as the initial angular momenta and label the remaining edges as the intermediate angular momenta of the left/right hand side of the recoupling coefficient. This can be done easily by traversing the defining tree handling first the children and afterwards the parent (postorder traversal). The result is a valid recoupling coefficient for which a summation formula can be generated.

In order to make a comparison, we generated a summation formula for each testcase with both `NEWGRAPH` and `RACAH` and with our algorithm `CCA-EC`. The

results are shown in Table 1. It is clear that the algorithm CCA-EC handles all cases without difficulty and that the summation formula generated by CCA-EC is always at least as good as the results obtained by NEWGRAPH and RACAH. Typically CCA-EC generates an equally complex formula for the small cases, but often, in particular for the larger cases, CCA-EC obtains a more efficient formula.

Another way to check the performance of the heuristic algorithm CCA-EC is an exhaustive search for the optimal reduction, which is feasible for small cases. For this purpose we used simplified versions of the reduction rules without explicit formula generation. The search performs a bounded depth-first traversal of the solution tree, taking into account all possible operations at each point, but giving preference to the operations the heuristic would choose. 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 [17] to prune branches not containing shortest reductions. A cheap minimum cost function is obtained from the following observations: (1) a Yutsis graph having a triangle can be reduced with one interchange if it is equal to K_4, 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 or $n = 5$) 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 [16], to find shorter reductions. The main idea behind LDS is that only a few of the decisions made by the heuristic are "wrong" in the search for a good reduction. For a solution tree of height d, there are only d ways that the heuristic could make one wrong decision 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 k 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 found some cases where LDS returns a better formula than CCA-EC, which means that in these cases 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-

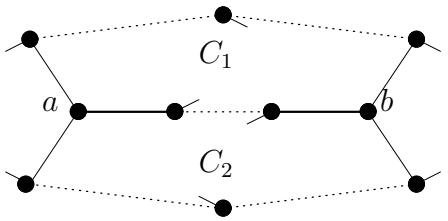


Fig. 6. Two cycles C_1, C_2 sharing a path from a to b. It is impossible to perform an interchange on the bold edges reducing both cycles in length.

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. Closer examination showed that one of the steps in LDS involves making a 6-cycle larger, while a 5-cycle is available, but none of the possible interchanges involving this 5-cycle leads to a reduction of 25 interchanges. Hence the interchange involving the 6-cycle was obviously a better thing to do at this point.

5 Heuristics with counting of cycles (`SB` and `CC`)

In many cases the heuristic `EC` reduces two cycles in length by performing only one interchange, since often 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 6. 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.

However, in such cases it is often still 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 2, will reduce a 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,

Case	(Cg_8)	(Cb_6)	(Cb_8)	$(Cg_{9,1})$	$(Cg_{9,2})$	$(Cg_{9,3})$	$(Cg_{9,4})$
CCA-EC	38	15	44	87	89	89	88
CCA-SB	37	14	42	86	83	85	85
CCA-CC	37	14	42	86	86	84	87
Case	$(Cg_{9,5})$	$(Cg_{9,6})$	$(Cg_{9,7})$	$(Cg_{9,8})$	$(Cg_{9,9})$	$(Cg_{9,10})$	$(Cg_{9,11})$
CCA-EC	89	92	90	90	91	93	90
CCA-SB	87	84	84	86	84	85	85
CCA-CC	86	85	84	85	83	85	85
Case	$(Cg_{9,12})$	$(Cg_{9,13})$	$(Cg_{9,14})$	$(Cg_{9,15})$	$(Cg_{9,16})$	$(Cg_{9,17})$	$(Cg_{9,18})$
CCA-EC	91	88	89	91	91	94	91
CCA-SB	83	84	86	84	85	87	86
CCA-CC	86	85	86	85	85	87	86

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

be the set of relevant cycles of length l which become smaller, resp. bigger, by performing some interchange I. 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 (SB): We prefer I if $S_l > S'_l$ or, when $S_l = S'_l$, if $B_l < B'_l$.

Cycle count (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 cycles, in which case both interchanges are considered equivalent. The time cost $h(n)$ of both heuristics is $O(n^{10})$, resulting in an overall time complexity of CCA-SB and CCA-CC of $O(n^{11} \log n)$.

In Table 2 we compare both heuristic algorithms CCA-SB and CCA-CC against CCA-EC which uses the heuristic based on edge costs. The cases labeled $Cg_{9,i}$, $i = 1, \ldots, 18$ are all 18 cubic cages of girth 9 having 58 nodes. On the cases dropped from Table 1 all heuristics returned a reduction with an equal amount of interchanges. It is interesting to note that both heuristics obtain the same result as CCA-EC for the case Cg_7, the case where LDS found a better reduction.

Since these cases are constructed to be hard, one could wonder if the gain of

these new heuristics is also reflected in 'more general' cases. For this reason we ran all three heuristic algorithms on several sets R_n of $100n$ randomly generated cubic graphs of size n, for $n = 6, \ldots, 30$ and $10n$ for $n = 50$. These random sets are generated with McKay's random regular graph generator `genrang`, a part of the popular `nauty` package for graph iso- and automorphisms [18]. For the small graph sizes ($6 \leq n \leq 25$) we filtered out the non-Yutsis graphs, including those having bridges, with the algorithm described in [15]. For the larger graphs ($n > 25$) we used an (unpublished) approximation algorithm, based on local search to find a defining tree, to confirm that all the graphs are Yutsis, after filtering out those having bridges with a linear time algorithm (see [9]). For this experiment we used again the simplified reduction rules without formula generation and consequently did not need to map a general recoupling coefficient on each graph. We collected the minimum, maximum and average number of interchanges together with the standard deviation for each random set. The results are shown in Table 3.

From these experiments it is clear that for small graphs the results obtained by `CCA-EC` are as good as those obtained by `CCA-SB` and `CCA-CC`, while for bigger graphs `CCA-SB` or `CCA-CC` often returns better results than `CCA-EC`. The experiments indicate a similar performance when comparing the heuristics `CCA-SB` or `CCA-CC`, with a little preference for `CCA-SB` depending from case to case.

6 Conclusion

Both heuristic algorithms `CCA-SB` and `CCA-CC` improve on the previous algorithms for generating an efficient analytical expression for general recoupling coefficients. Certainly for applications that need to generate a summation formula for complicated cases once and evaluate it several times, the heuristics `SB` and `CC` are the best choice. For applications which require a lot of resources and where general recoupling coefficients are immediately evaluated after generation of the summation formula, `CCA-EC` would be more suitable due to its better time complexity ($O(n^6 \log n)$ vs. $O(n^{11} \log n)$). For cases with $n \leq 5$ there is no need to use the heavier heuristics, since the `EC` heuristic is confirmed to be optimal for these small cases.

From the experiments we also know that none of the heuristics is optimal, since we found shorter reductions using LDS. An even better heuristic should take into account the effect of cycles sharing paths in an anticipating manner in order to avoid as much as possible that applying the reduction rules would lead to such a situation. Using LDS we also found a counter-example showing that reducing the smallest cycle first does not always lead to a shortest reduction and thus to a most efficient summation formula.

		CCA-EC				CCA-SB				CCA-CC			
n	$\lvert R_n\rvert$	min	max	μ	s_{n-1}	min	max	μ	s_{n-1}	min	max	μ	s_{n-1}
6	591	2	9	5.91	1.1	2	9	5.91	1.1	2	9	5.91	1.1
7	689	2	11	7.60	1.1	2	11	7.60	1.2	2	11	7.60	1.2
8	793	4	13	9.38	1.4	4	13	9.36	1.6	4	13	9.36	1.6
9	896	6	16	11.22	1.7	6	16	11.18	1.7	6	16	11.18	1.4
10	995	6	18	13.22	1.8	6	18	13.12	1.7	6	18	13.12	1.7
11	1095	9	24	15.18	2.1	8	22	15.06	2.1	8	22	15.06	2.0
12	1197	9	24	17.41	2.4	9	23	17.23	2.3	9	23	17.23	2.4
13	1297	11	28	19.63	2.6	11	26	19.38	2.4	11	26	19.38	2.4
14	1397	12	30	22.22	2.7	12	30	21.87	2.6	12	29	21.88	2.5
15	1499	16	33	24.68	2.8	16	32	24.25	2.6	16	32	24.26	2.5
16	1599	16	37	27.08	2.9	16	35	26.58	2.7	16	35	26.59	2.8
17	1699	19	39	29.72	3.1	19	37	29.10	2.9	19	38	29.11	2.9
18	1799	20	42	32.33	3.3	20	41	31.60	3.0	20	40	31.63	3.0
19	1897	22	46	35.21	3.4	22	44	34.37	3.1	22	44	34.38	3.1
20	2000	24	48	37.98	3.7	23	47	36.97	3.3	23	47	37.00	3.3
21	2095	25	51	40.76	3.7	24	51	39.68	3.5	24	51	39.70	3.5
22	2199	28	55	43.69	3.8	28	52	42.46	3.5	28	52	42.48	3.5
23	2298	29	58	46.76	3.9	29	55	45.36	3.5	29	56	45.39	3.6
24	2398	34	62	49.45	4.3	34	59	47.95	3.8	34	59	47.97	3.8
25	2500	35	65	52.61	4.3	35	63	50.95	3.8	35	62	50.98	3.8
26	2599	39	70	55.62	4.3	40	66	53.81	3.9	40	66	53.86	4.0
27	2697	42	73	58.79	4.6	41	68	56.84	4.2	41	69	56.85	4.2
28	2800	46	75	62.04	4.5	45	73	59.94	4.1	45	72	59.98	4.2
29	2900	48	80	65.29	4.8	44	75	63.01	4.3	44	76	63.04	4.3
30	2999	51	83	68.56	5.0	51	79	66.15	4.5	51	79	66.19	4.5
50	500	118	159	139.79	6.5	114	152	133.93	6.0	114	149	133.93	5.9

Table 3
Comparison of minimum, maximum and average number of interchanges performed by CCA-EC, CCA-SB and CCA-CC together with the standard deviation on randomly generated sets R_n of cubic graphs with $2n$ nodes and $3n$ edges.

7 Acknowledgment

We wish to thank our colleague Stijn Lievens for several enlightening discussions considering the physical background of the problem. We also would like to thank the anonymous referees, who gave very helpful comments on the first version of this article.

References

[1] 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).

[2] A. Bar-Shalom, M. Klapisch, *Computer Physics Communications* **50** (1988) 375.

[3] P. M. Lima, *Computer Physics Communications* 66 (1991) 89.

[4] V. Fack, S. N. Pitre and J. Van der Jeugt, *Computer Physics Communications* **101** (1997) 155.

[5] S. Fritzsche, T. Inghoff, T. Bastug, M. Tomaselli, *Computer Physics Communications* **139** (2001) 314.

[6] 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).

[7] D. Van Dyck, V. Fack, *Computer Physics Communications* **151** (2003) 354–368.

[8] D. Van Dyck, V. Fack, *Computer Physics Communications* **154** (2003) 219–232.

[9] R. Tarjan, *SIAM Journal of Computing* **1** (1972) 146–160.

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

[11] V. Fack, S. Lievens, J. Van der Jeugt, *Computer Physics Communications* **119** (1999) 99–114.

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

[13] V. Fack, S. N. Pitre, J. Van der Jeugt, *Computer Physics Communications* **83** (1994) 275.

[14] G. Brinkmann, "Generating cubic graphs faster than isomorphism checking", preprint SFB 343, No. 92-047, Bielefeld (1993).

[15] D. Van Dyck, V. Fack, "A fast algorithm for filtering Yutsis graphs", submitted to *Journal of Graph Algorithms and Applications* (2003).

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

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

[18] B. McKay, nauty. `http://cs.anu.edu.au/people/bdm/nauty`.