An efficient algorithm for the Riemannian 10j symbols

J Daniel Christensen and Greg Egan

1 Department of Mathematics, University of Western Ontario, London, Ontario, Canada
E-mail: jdc@uwo.ca and gregegan@netspace.net.au

Received 30 November 2001
Published 6 March 2002
Online at stacks.iop.org/CQG/19/1185

Abstract
The 10j symbol is a spin network that appears in the partition function for the Barrett–Crane model of Riemannian quantum gravity. Elementary methods of calculating the 10j symbol require $O(j^9)$ or more operations and $O(j^2)$ or more space, where $j$ is the average spin. We present an algorithm that computes the 10j symbol using $O(j^5)$ operations and $O(j^2)$ space, and a variant that uses $O(j^6)$ operations and a constant amount of space. An implementation has been made available on the web.

PACS numbers: 0240K, 0460, 0460N

1. Introduction

The Barrett–Crane model of four-dimensional Riemannian quantum gravity [6] has been of significant interest recently [1, 2, 10, 12]. The model is discrete and well defined, and the partition function for the Perez–Rovelli version has been rigorously shown to converge [11] for a fixed triangulation of spacetime. The Riemannian model serves as a step along the way to understanding the less tractable but physically more realistic Lorentzian version [7]. However, despite its simplicity, we are currently lacking explicit numerical computations of the partition function and of expectation values of observables in the Riemannian model. These are necessary to test its large-scale behaviour and other physical properties.

It has been shown [3] that the amplitudes in the Barrett–Crane model are always non-negative, and therefore the expectation values of observables can be approximated using the Metropolis algorithm. This greatly reduces the number of samples that must be taken, and thus the remaining obstacle is the time required to compute each sample. This paper presents a very efficient algorithm for doing these computations. The algorithm is used in [4] and [5] to understand the asymptotic behaviour of the 10j symbols and the dependence of the partition function on a cut-off.

To explain further, we need to describe the Barrett–Crane model in more detail. It has been formulated by Baez [2] as a discrete spin foam model, in which faces in the dual 2-skeleton of a fixed triangulation of spacetime are labelled by spins. The dual 2-skeleton
consists of a dual vertex at the centre of each 4-simplex of the triangulation, five dual edges incident to each dual vertex (one for each tetrahedron in the boundary of the 4-simplex), and ten dual faces incident to each dual vertex (one for each triangle in the boundary of the 4-simplex).

Baez notes that the partition function for this model is the sum, over all labellings of the dual faces by spins, of an expression that contains the product of a $10j$ symbol for each dual vertex. A $10j$ symbol, described in detail in section 2, is a Spin(4) spin network. Roughly speaking, a spin network is a graph whose vertices are labelled by tensors, and whose edges indicate how to contract these tensors. A spin network evaluates to a complex number in the way explained in section 3. In short, the $10j$ symbol is a function taking ten input spins and producing a complex number. It is at the heart of the calculation of the partition function, and thus an algorithm for calculating the $10j$ symbols efficiently is quite important.

In section 2, we recall the definition of the $10j$ symbol using spin networks. Then in section 3, we briefly describe the elementary algorithms for evaluating these spin networks, and give their running times and memory use. We conclude with section 4, which presents our algorithms and their time and space needs.

2. The $10j$ symbol

In the dual 2-skeleton of a triangulation of a 4-manifold, each dual vertex belongs to five dual edges, and each pair of these dual edges borders a dual face. A $10j$ symbol is a Spin(4) spin network with five vertices (corresponding to the five dual edges) and ten edges, one connecting each pair of vertices (corresponding to the ten dual faces), with the edges labelled by spins. In the context of a Spin(4) spin network, a spin $j$ labelling an edge denotes the representation $j \otimes j$ of Spin(4) $\cong SU(2) \times SU(2)$, where $j$ is the spin-$j$ representation of $SU(2)$. Such representations are called ‘balanced.’ We use the convention that spins are non-negative half-integers.

Here is a diagram of the $10j$ symbol, with the vertices numbered 0 through 4, and the spins divided into two groups: $j_{1,i}$ are the spins on the edges joining vertex $i$ to vertex $i + 1$ (modulo 5), and $j_{2,i}$ are the spins on the edges joining vertex $i$ to vertex $i + 2$ (modulo 5).

\begin{center}
\begin{tikzpicture}
\node[circle,draw,inner sep=3pt] (0) at (0,0) {$0$};
\node[circle,draw,inner sep=3pt] (1) at (1,2) {$1$};
\node[circle,draw,inner sep=3pt] (2) at (-1,2) {$2$};
\node[circle,draw,inner sep=3pt] (3) at (0,-2) {$3$};
\node[circle,draw,inner sep=3pt] (4) at (1,-2) {$4$};
\draw[thick] (0) -- (1) node[above] {$j_{1,0}$};
\draw[thick] (0) -- (2) node[above] {$j_{2,0}$};
\draw[thick] (0) -- (3) node[below] {$j_{1,1}$};
\draw[thick] (0) -- (4) node[below] {$j_{2,1}$};
\draw[thick] (1) -- (2) node[right] {$j_{1,2}$};
\draw[thick] (1) -- (3) node[below] {$j_{2,2}$};
\draw[thick] (1) -- (4) node[above] {$j_{1,3}$};
\draw[thick] (2) -- (3) node[below] {$j_{2,3}$};
\draw[thick] (2) -- (4) node[above] {$j_{1,4}$};
\end{tikzpicture}
\end{center}
The five vertices of the network are equal to Barrett–Crane intertwiners. These are the unique intertwiners (up to a factor) between four balanced representations of Spin(4) with the property that their expansion as a sum of tensor products of trivalent SU(2) networks only contains balanced representations on the internal edge, regardless of which pairs of external edges are joined [13]. Barrett and Crane give the formula for these intertwiners in [6]:

\[
\begin{align*}
\text{\( j_1 \) \& \( j_2 \) \& \( j_3 \) \& \( j_4 \))} & := \sum_l \Delta_l \cdot \\
\text{\( j_1 \) \& \( j_2 \) \& \( j_3 \) \& \( j_4 \))}
\end{align*}
\]

Here the sum is over all admissible values of \( l \), i.e. those that satisfy the Clebsch–Gordan condition for both SU(2) vertices. So \( l \) ranges from \( \max(|j_1 - j_2|, |j_3 - j_4|) \) to \( \min(j_1 + j_2, j_3 + j_4) \) in integer steps. If the difference between these bounds is not an integer, the Spin(4) vertex will be zero. When \( l \) satisfies these conditions, there is a unique intertwiner up to normalization which can be used to label the trivalent SU(2) vertices. (These intertwiners are normalized so that the theta network in the numerator of equation (4) has value 1.) \( \Delta_l \) is the value of a loop in the spin-\( l \) representation, which is just \((-1)^{2l} (2l + 1)\), the superdimension of the representation.

The uniqueness result of Reisenberger [13] tells us that if we replace the vertical edges in the above definition by horizontal edges, the result differs at most by a constant factor. In fact, Barrett and Crane [6] stated that the two definitions give exactly the same Spin(4) vertex and Yetter [14] has proved this.

Any closed spin network evaluates to a complex number by contracting the tensors at the vertices according to the pairings specified by the edges. Thus the 10\(j\) symbol is a complex number. (In fact, one can show that it is always a real number.)

To avoid confusion, we want to make it clear that we are working with the classical (non-\( q \)-deformed) evaluation of our spin networks. We will frequently reference the book by Kauffman and Lins [9]; while it explicitly discusses the \( q \)-deformed version, the formulae we use apply to the classical evaluation as well.

### 3. Elementary algorithms

To set the context, we begin by explaining some elementary algorithms for computing the 10\( j \) symbol. These algorithms all share the feature that they evaluate a spin network by choosing bases for the representations labelling the edges, computing the components of the tensors representing the intertwiners, and computing the contraction of the tensors in some way. They make no use of special features of these tensors, except for the vanishing property mentioned below.

The first three methods each have two versions, one which works directly with the Spin(4) network (1), and one that converts it into a five-fold sum over SU(2) networks, by expanding each Barrett–Crane intertwiner:
Here, \( l_i \) is the spin labelling the new edge introduced by the expansion of the intertwiner at vertex \( i \), and it ranges in integer steps from \( L_i := \max(|j_{1,i} - j_{2,i}|, |j_{1,i} - j_{2,i-2}|) \) to \( H_i := \min(j_{1,i} + j_{2,i}, j_{1,i-1} + j_{2,i-2}) \), where the vertex numbers are all to be interpreted modulo 5. (If \( H_i - L_i \) is not a non-negative integer, then the sum over \( l_i \) is empty and the 10 \( j \) symbol is zero. In fact, if vertex \( i \) is non-zero, then \( |j_{1,i} - j_{2,i}|, |j_{1,i} - j_{2,i-2}|, j_{1,i} + j_{2,i} \) and \( j_{1,i-1} + j_{2,i-2} \) must all differ by integers.) There are at most \( O(j) \) terms in each sum, where \( j \) is the average of the ten spins. Since the two decagonal networks are the same, one only needs to evaluate one of them and square the answer.

The first elementary method is one we call direct contraction. One simply labels each edge in the spin network with a basis vector from the representation labelling the edge, and multiplies together the corresponding components of the tensors. This is then summed up over all labellings. In fact, one can restrict to a smaller set of labellings: the bases can be chosen so that for each choice of two basis vectors on two of the three edges meeting an \( SU(2) \) vertex, there is at most one choice of basis vector on the third edge giving a non-zero tensor component. The \( Spin(4) \) vertices also have the property that the bases can be chosen so that when three of the basis vectors adjacent to a vertex are specified, the last one is determined.

The second elementary method is staged contraction. In the \( Spin(4) \) version of this method, one starts with the tensor at vertex 0, contracts with the tensor at vertex 1, and then vertex 4, vertex 2, and finally vertex 3, again taking care to save space and time by using the vanishing properties of the tensors. Similarly, one can iteratively contract the tensors in the decagonal \( SU(2) \) network. At intermediate stages, one is storing tensors with a large number of components.

The third elementary method is 3cut. Here one takes a ray from the centre of (1) and cuts the three edges it crosses. Then one takes the trace of the operator this defines on the three-fold tensor product. In more detail, one sums over basis vectors for the factors in this tensor product, computing the effect of the network on these basis vectors, and using the vanishing properties. The memory required for this method (and the next) is dominated by the memory needed to store the tensors themselves.

The fourth and final elementary method is 2cut. This one only makes sense for the decagonal network since one proceeds by taking a ray from the centre of the decagon which crosses just two edges, cutting those two edges, and taking the trace of the resulting operator, using the vanishing properties.

Here is a table which gives an upper bound on the number of operations (additions and multiplications) that these algorithms use, and the amount of memory they require, as a function of a typical spin \( j \). The space requirements include the space to store the Barrett–Crane tensors:
An efficient algorithm for the Riemannian 10j symbols

|          | Direct contraction | Staged contraction | 3cut | 2cut |
|----------|--------------------|--------------------|------|------|
| Spin(4)  | $j^{12}$           | $j^{12}$           | $j^{12}$ N/A |
| Space    | $j^6$              | $j^{10}$           | $j^6$ N/A |
| SU(2)    | $j^{11}$           | $j^9$              | $j^{10}$ $j^9$ |
| Space    | $j^2$              | $j^4$              | $j^2$ $j^2$ |
| Either   | $mp^6$             | $mp^{2v+4}$        | $mp^{v+5}$ $mp^4$ |
| Space    | $p^{v+2}$          | $p^{v+4}$          | $p^{v+2}$ $p^2$ |

In the last two rows, we represent the entries in a uniform way for either version by writing $v = 1$ for the Spin(4) version of each algorithm and $v = 0$ for the SU(2) version. Then we let $p = j^{v+1}$ and $m = j^{3-5v}$. This shows how they are related. For example, the SU(2) methods always get a factor of $j^2$ in time from the five loops generated by expanding the Barrett–Crane vertices. Also, the Spin(4) methods get powers of $j^2$ (because $\dim j \otimes j = (2j+1)^2$), while the SU(2) methods get powers of $j$ (because $\dim j = (2j+1)$).

The 2cut method has the best worst-case behaviour, in both space and time.

In the next section we present an algorithm which has running time $O(j^5)$ and requires $O(j^2)$ space, and give variants with running time $O(j^6)$ and $O(j^5)$ and which use a constant amount of space.

The difference between $j^5$ and the running time, $j^6$, for the best of the elementary methods is significant. For example, with all spins equal to 20, our $j^5$ algorithm runs in under six minutes on a 300 MHz microprocessor. A back of the envelope calculation suggests that this would take about 30 years with a $j^6$ algorithm.

### 4. Our new algorithms

In this section we describe a new method for computing 10j symbols. The key feature of this method is that it does not proceed by computing the tensor components for the intertwiners. Instead, it uses recoupling to simplify the network to one that can be evaluated directly. Thus this method makes use of special properties of the tensors that occur in the 10j symbols.

There are three versions of this method. We explain one of these in some detail, and briefly describe the variants at the appropriate points. We use the notation of equation (1), and consider the expansion (2) as a sum of squares of decagonal networks.

The decagonal networks can be deformed into the ‘ladders’ shown below, where the vertices at the bottom of each ladder are to be identified with those at the top of the same ladder. Any sign introduced by this deformation is produced twice and so can be ignored:
To simplify these networks further, we recouple the sub-networks consisting of a horizontal edge and half of each vertical edge incident to its endpoints, rewriting them as sums of sub-networks with the same external edges, using the following recoupling formula for SU(2) spin networks [9, ch 7):

\[
\{a \ b \ k\} \ {c \ d \ j\} = \sum_k \{a \ b \ k\} \ {c \ d \ j\}
\]

The horizontal and vertical networks appearing above are two different ways of writing intertwiners from one two-fold tensor product of irreducible representations to another; both kinds of networks form bases for the space of intertwiners, and the 6j symbols appearing in the formula are defined to be the change-of-basis coefficients.

At first glance, it might look as if this recoupling would introduce sums over ten new spins, labelling the ten new vertical edges. However, the total networks that result from the recoupling consist of chains of sub-networks shaped as

\[
\begin{array}{ccc}
  a & b & c \\
  j & j & j \\
  d & d & d \\
\end{array}
\]

By Schur’s lemma, such sub-networks will only be non-zero when the incoming and outgoing edges have identical spins. So the recoupled networks can be written as a sum over just two new spins, \(m_1\) and \(m_2\):

\[
\sum_{l_0, \ldots, l_4} \prod_{k=0}^4 \Delta_{b_k} \sum_{m_1, m_2} \left\{ \begin{array}{ccc}
  j_{2,0} & l_0 & m_1 \\
  j_{2,4} & l_1 & j_{1,0} \\
  j_{2,4} & l_1 & j_{1,1} \\
  j_{2,0} & m_1 & l_1 \\
  j_{2,1} & j_{1,1} & j_{1,2} \\
  j_{2,2} & l_2 & m_1 \\
  j_{2,1} & j_{1,2} & j_{1,3} \\
  j_{2,2} & l_3 & j_{1,3} \\
  j_{2,4} & l_4 & m_1 \\
  j_{2,3} & j_{1,4} & j_{1,3} \\
\end{array} \right\} 
\]

This sum is over all values such that every vertex in the diagram satisfies the Clebsch–Gordan condition, so both \(m_1\) and \(m_2\) will independently range in integer steps from \(\max_i (|l_i - j_{2,i-1}|)\) to \(\min_i (l_i + j_{2,i-1})\). Here we use the fact that if the five Barrett–Crane vertices are non-zero, then as \(i\) varies, the ten quantities \(|l_i - j_{2,i-1}|\) and \(l_i + j_{2,i-1}\) all differ by integers.
Indeed, \(|l_i - j_{2,i-1}| \equiv l_i - j_{2,i-1} \equiv l_i + j_{2,i-1}\) modulo integers, and by the paragraph after equation (2),

\[ l_i + j_{2,i-1} \equiv |j_{1,i} - j_{2,i}| + j_{2,i-1} \equiv j_{1,i} + j_{2,i-1} + j_{2,i} \equiv j_{1,i+1} + j_{2,i+1} + j_{2,i} \equiv l_{i+1} + j_{2,i} \]

modulo integers, where in the third step we use, the vertex \(i + 1\) is non-zero.

At this point there is a choice which determines which version of the algorithm one obtains. If the sum over \(m_1\) and \(m_2\) is left inside the sum over the \(l_i\), then it can be written as the square of a sum over a single \(m\). As described below, each of the terms in this sum can be computed with \(O(j^7)\) operations and a constant amount of memory, where \(j\) is the average of the ten spins. Thus, this method produces an algorithm that runs in \(O(j^7)\) time and takes a constant amount of space.

In general, it turns out to be more efficient to make the sum over the \(m\) values outermost, in order to reinterpret the sum over the \(l_i\) as the trace of a matrix product. The range for \(m_1\) and \(m_2\) must encompass all potentially admissible values, and the range for each \(l_i\) can then be adjusted for the current values of \(m_1\) and \(m_2\). The original range for \(l_i\) was from \(L_i\) to \(H_i\) (see the paragraph after equation (2)), so the \(m\) values can never be greater than \(\min(H_i, j_{2,i-1})\) without violating the triangle inequality at one of the vertices. The lower bound is given by \(\max(\min_l(|l_i - j_{2,i-1}|; L_i \leq l_i \leq H_i))\), where the minimum breaks down into three cases: if \(j_{2,i-1} \geq H_i\), it is \(j_{2,i-1} - H_i\); if \(j_{2,i-1} \leq L_i\), it is \(L_i - j_{2,i-1}\); and if \(L_i < j_{2,i-1} < H_i\), it is either 0 or \(\frac{1}{4}\), depending on whether \(2j_{2,i-1} \equiv 2L_i \pmod{2}\) or not.

Each \(l_i\) is then restricted to take account of the current \(m\) values, ranging from \(\max(L_i, |m_1 - j_{2,i-1}|, |m_2 - j_{2,i-1}|)\) to \(\min(H_i, m_1 + j_{2,i-1}, m_2 + j_{2,i-1})\).

By Schur’s lemma, each sub-network of the form (3) is equal to a multiple of the identity, so each of the recoupled ladders, with top and bottom edges joined, is simply a multiple of a loop. Kauffman and Lins [9] give the following formula, which can be checked by taking the trace of both sides:

\[
\begin{array}{ccc}
\theta(a, b, c) &=& \frac{\text{Tet}(a, b, c, i) \Delta_i}{\theta(a, d, i) \theta(b, c, i)}
\end{array}
\]

Equations (4) and (5) allow us to write the following expression for the 10j symbol:

\[
\sum_{m_1, m_2} (2m_1 + 1)(2m_2 + 1)(-1)^{2i(H_i, j_{2,i-1}) - m_1 - m_2} \prod_{l_0, \ldots, l_4} (M_{k_1}^{m_1, m_2})_{l_0}^{l_{k+1}}
\]

where

\[
(M_{k_1}^{m_1, m_2})_{l_0}^{l_{k+1}} = \frac{\Delta_{l_0} \text{Tet}(l_{k+1}, j_{2,k}, m_1) \text{Tet}(l_{k+1}, j_{2,k}, m_2)}{\theta(j_{2,k}, l_{k+1}, m_1) \theta(j_{2,k}, l_{k+1}, m_2)}.
\]
The matrices become equation (7) to be unity and normalize the tetrahedral networks. Including this normalization, the matrix elements by the appropriate \( \theta \) scale at a lower power than the convention used in the formula for the Barrett–Crane intertwiner. Each of the \( O(\text{intertwiners}) \) introduce signs of \( \pm \theta \) for the tetrahedral and \( O(\text{networks}) \) is always unity. We have made use of the symmetries of the tetrahedral networks to put the coefficients in a uniform order for all terms. The sum over the \( l_i \) in equation (6) is the trace of the product of the five matrices \( M^{m_1,m_2}_k \). For each pair of values of \( m_1 \) and \( m_2 \), these matrices can be computed using closed formulae for the tetrahedral and \( \theta \) networks given by Kauffman and Lins in [9]. The formula for the tetrahedral networks involves a sum with \( O(j) \) terms, so computing each matrix requires \( O(j^5) \) operations\(^1\). The trace of the matrix product can also be found in \( O(j^2) \) steps. There are two factors of \( j \) coming from the sums over \( m_1 \) and \( m_2 \), yielding an overall count of \( O(j^7) \) operations. This method requires \( O(j^2) \) space to store the matrices.

For some 10-tuples of spins, if all the spins are multiplied by \( \lambda \), the time required will scale at a lower power than \( \lambda^5 \). Multiplying all the spins by a factor will increase the upper and lower bounds of all the sums linearly, but in cases where the two bounds are equal, the sum will consist of a single term, regardless of the scaling factor. When many of the upper and lower bounds of all the sums linearly, but in cases where the two bounds are equal, the sum becomes faster than the \( O(j^5) \) version. Thus one may wish to use the first variant for certain 10\( j \) symbols.

For large spins, the memory usage can be a problem. For example, with spins of around 180, storing each matrix \( M^{m_1,m_2}_k \) requires about 1 gigabyte. In this case, one can recalculate the matrix entries as needed, resulting in \( O(j^6) \) time and \( O(j^4) \) space (\( O(j^4) \) if factorials are cached).

The formulae in [9] for the network evaluations are unnormalized. To normalize all the \( SU(2) \) intertwiners according to the convention that any \( \theta \) network has a value of 1—which is the convention used in the formula for the Barrett–Crane intertwiner—it is simpler to divide the matrix elements by the appropriate \( \theta \) networks than to take the existing \( \theta \) networks in equation (7) to be unity and normalize the tetrahedral networks. Including this normalization, the matrices become

\[
\left( N^k_{m_1,m_2} \right)^{l_{\text{int}}} = \frac{\left( M^{m_1,m_2}_k \right)^{l_{\text{int}}}}{\theta( j_2,k-1, l_{k+1}, j_{l,k}) \theta( j_2,k+1, l_{k+1}, j_{l,k+1})} \quad (8)
\]

A subroutine written in C++ that implements this algorithm is available on the web [8], along with some sample computations.

We have not dealt explicitly with the \( q \)-deformed case, where the representations of \( SU(2) \) are replaced with representations of \( SU(2)_q \), but it is straightforward to adapt each stage of the development above using the formulae in [9] for the \( q \)-deformed twist, loop, \( \theta \) and tetrahedral networks.

**Acknowledgment**

The authors would like to thank J Baez for many useful conversations about the material in this paper.

---

\(^1\) Each of the \( O(j) \) terms in the formula for the tetrahedral network contains factorials, which themselves require \( O(j) \) operations. However, with some care, the formula can be evaluated with a total of \( O(j) \) operations. In practice, we precalculate the factorials using \( O(j) \) space.
References

[1] Baez J C 1998 Spin foam models Class. Quantum Grav. 15 1827–58
[2] Baez J C 2000 An introduction to spin foam models of quantum gravity and BF theory Geometry and Quantum Physics ed H Gausterer and H Grosse (Berlin: Springer) (Baez J C 1999 Preprint gr-qc/9908087)
[3] Baez J C and Christensen J D 2001 Positivity of spin foam amplitudes Preprint gr-qc/0110044 (Class. Quantum Grav: at press)
[4] Baez J C, Christensen J D and Egan G Asymptotics of 10j symbols in preparation
[5] Baez J C, Christensen J D, Halford T R and Tsang D C 2002 Spin foam models of Riemannian quantum gravity Preprint gr-qc/0202017
[6] Barrett J and Crane L 1998 Relativistic spin networks and quantum gravity J. Math. Phys. 39 3296–302 (Barrett J and Crane L 1997 Preprint gr-qc/9709028)
[7] Barrett J W and Crane L 2000 A Lorentzian signature model for quantum general relativity Class. Quantum Grav. 17 3101–18
[8] Christensen J D Spin foams page http://jdc.math.uwo.ca/spin-foams/
[9] Kauffman L and Lins S 1994 Temperley–Lieb Recoupling Theory and Invariants of 3-Manifolds (Princeton, NJ: Princeton University Press)
[10] Oriti D 2001 Spacetime geometry from algebra: spin foam models for non-perturbative quantum gravity Preprint gr-qc/0106091
[11] Perez A 2001 Finiteness of a spin foam model for Euclidean quantum general relativity Nucl. Phys. B 599 427–34
[12] Perez A and Rovelli C 2001 A spin foam model without bubble divergences Nucl. Phys. B 599 255–82
[13] Reisenberger M P 1999 On relativistic spin network vertices J. Math. Phys. 40 2046–54 (Reisenberger M P 1998 Preprint gr-qc/9809067)
[14] Yetter D N 1999 Generalized Barrett–Crane vertices and invariants of embedded graphs J. Knot Theory Ramifications 8 815–29 (Yetter D N 1998 Preprint math.QA/9801131)