```
GSWalk* PatternTree::expand (pair<float, string> max, const int parent_size) {

  assert(parent_size>0);

  fm::statistics->patternsize++;
  if ( fm::statistics->patternsize > (int) fm::statistics->frequenttreenumbers.size () ) {
    fm::statistics->frequenttreenumbers.resize ( fm::statistics->patternsize, 0 );
    fm::statistics->frequentpathnumbers.resize ( fm::statistics->patternsize, 0 );
    fm::statistics->frequentgraphnumbers.resize ( fm::statistics->patternsize, 0 );
  }
  ++fm::statistics->frequenttreenumbers[fm::statistics->patternsize-1];
  if ( fm::statistics->patternsize == ((1<<(sizeof(NodeId)*8))-1) ) {
    fm::statistics->patternsize--;
    return NULL;
  }


  // new siblingwalk
  GSWalk* siblingwalk = new GSWalk();

  // needed for topdown and sibling merge
  vector<int> core_ids;
  for (int j=0; j<parent_size; j++) core_ids.push_back(j);

  for ( int i=legs.size()-1; i>=0; i-- ) {


    // new current pattern
    GSWalk* gsw = new GSWalk();
    GSWalk* topdown = NULL;

    bool nsign=1;

    if (fm::chisq->active) fm::chisq->Calc(legs[i]->occurrences.elements);
    float cur_chisq=fm::chisq->p;

    fm::graphstate->insertNode ( legs[i]->tuple.connectingnode, legs[i]->tuple.label, legs[i]-
>occurrences.maxdegree );
    #ifdef DEBUG
    fm::graphstate->print(legs[i]->occurrences.frequency);
    #endif

    #ifdef DEBUG
    fm::gsp_out=false;
    string s = fm::graphstate->to_s(legs[i]->occurrences.frequency);
    bool diehard=0;
    //if (s.find("N-C-C(-O-C-N)(=C-C)")!=string::npos) { fm::die=1; diehard=1; }
    #endif

    if (fm::chisq->active) {
        map<Tid, int> weightmap_a; each_it(fm::chisq->fa_set, set<Tid>::iterator) {
weightmap_a.insert(make_pair((*it),1)); }
        map<Tid, int> weightmap_i; each_it(fm::chisq->fi_set, set<Tid>::iterator) {
weightmap_i.insert(make_pair((*it),1)); }
        fm::graphstate->print(gsw, weightmap_a, weightmap_i); // print to graphstate walk
        gsw->activating=fm::chisq->activating;
        if (cur_chisq >= fm::chisq->sig) nsign=0;
    }
    const int gsw_size = gsw->nodewalk.size();
```

```
    // !STOP: MERGE TO SIBLINGWALK
    if (gsw->to_nodes_ex.size() || siblingwalk->to_nodes_ex.size()) { cerr<<"Error! Already nodes marked
    as available 5.1. "<<gsw->to_nodes_ex.size()<<" "<<siblingwalk->to_nodes_ex.size()<<endl;exit(1); }
    if (nsign || gsw->activating!=siblingwalk->activating) { // empty sw needs no checks
          if (siblingwalk->hops>1) {
                siblingwalk->svd();
                cout << siblingwalk ;
          }
          delete siblingwalk;
          siblingwalk = new GSWalk();
    }
    if (!nsign && ((gsw->activating==siblingwalk->activating) || !siblingwalk->edgewalk.size())) {
        #ifdef DEBUG
        if (fm::die) cout << "CR gsw" << endl;
        #endif
        int res=gsw->conflict_resolution(core_ids, siblingwalk);
    }

    if (gsw->to_nodes_ex.size() || siblingwalk->to_nodes_ex.size()) { cerr<<"Error! Still nodes marked
    as available 5.1. "<<gsw->to_nodes_ex.size()<<" "<<siblingwalk->to_nodes_ex.size()<<endl; exit(1); }


    // RECURSE
    if ( ( !fm::do_pruning ||  (  fm::chisq->u >= fm::chisq->sig) ) &&
         (  fm::refine_singles || (legs[i]->occurrences.frequency>1) )
       ) {
        PatternTree p ( *this, i );
        if (cur_chisq > max.first) { fm::updated = true; topdown = p.expand (pair<float,
        string>(cur_chisq,fm::graphstate->to_s(legs[i]->occurrences.frequency)), gsw_size); }
        else topdown = p.expand (max, gsw_size);
    }

    // merge to siblingwalk
    if (topdown != NULL) {
       if (topdown->edgewalk.size()) {

            #ifdef DEBUG
            if (fm::die) {
                cout << "TOPDOWN2 BEGIN " << core_ids.size() << endl;
                cout << topdown ;
                cout << "--result--" << endl;
                cout << siblingwalk ;
            }
            #endif

            if (topdown->to_nodes_ex.size() || siblingwalk->to_nodes_ex.size()) { cerr << "Error!
            Already nodes marked as available 5.2. " << topdown->to_nodes_ex.size() << " " <<
            siblingwalk->to_nodes_ex.size() <<  endl; exit(1); }
            // STOP: OUTPUT TOPDOWN
            if (nsign || siblingwalk->activating!=topdown->activating) {
                #ifdef DEBUG
                if (fm::die) cout << "STOP CRITERIUM at CHI " << cur_chisq << endl;
                #endif
                if (topdown->hops>1) {
                    topdown->svd();
                    cout << topdown;
                }
            }
```

```
            // ELSE: MERGE TO SIBLINGWALK
            else {
                int res=topdown->conflict_resolution(core_ids, siblingwalk);
            }
            if (topdown->to_nodes_ex.size() || siblingwalk->to_nodes_ex.size()) { cerr << "Error! Still
            nodes marked as available 5.2. " << topdown->to_nodes_ex.size() << " " << siblingwalk-
            >to_nodes_ex.size() <<  endl; exit(1); }

            #ifdef DEBUG
            if (fm::die) {
                cout << "TOPDOWN2 END " << core_ids.size() << endl;
                cout << topdown ;
                cout << "--result--" << endl;
                cout << siblingwalk ;
            }
            #endif

        }
    }

    fm::graphstate->deleteNode ();
    delete topdown;
    delete gsw;
    #ifdef DEBUG
    if (diehard==1) {
       cerr << "DYING HARD!" << endl;
       exit(0);
    }
    #endif

  }

  #ifdef DEBUG
  if (!legs.size()) cout << fm::graphstate->sep() << endl;
  #endif

  fm::statistics->patternsize--;
  return siblingwalk;

}
```