Case Study II: Multi-query Roadmap Planners: Building C-Space Distance Oracles

Kostas Bekris
Computer Science and Engineering
University of Nevada, Reno

September 26, 2008
IROS 2008, Nice, France
Overview

Roadmap motion planners
- Preprocess the configuration space
- Efficient in answering multiple queries

This presentation
- OOPSMP’s version of PRM
  - Study the implementation
- How to extend PRM and design new techniques
  - Visibility-PRM
  - Spanner Roadmaps

Motivation

Roadmaps can be used as C-space preprocessing tools in problems beyond traditional motion planning
Probabilistic Roadmap Planner

[Kavraki, Svestka, Overmars, Latombe ‘96]

C-space  Sampling  Roadmap
Probabilistic Roadmap Planner

• Important Elements - Data Structures in OOPSMP implementation:
  – Inner classes
    • EdgeData: Edge cost, Local path, etc.
    • VertexData: State, Node id, etc.
    • PathIterator
    • StatelIterator
  – A roadmap: \textit{Graph}_t \text{ \textit{m}} \_ \textit{roadmap}
  – A disjoint set: \textit{DisjointSet} < \text{int} > \* \text{ \textit{m}} \_ \textit{disjointSet}
  – Search graph: \textit{SearchGraph}_t \text{ \textit{m}} \_ \textit{searchGraph}
  – Proximity Info: \textit{ProximityKeyPointer}_t \text{ \textit{m}} \_ \textit{proximity}
    \textit{ProximityQuery}<\text{OOPSMPpointer}_t> \text{ \textit{m}} \_ \textit{proximityQuery}
    \textit{ProximityResults}<\text{OOPSMPpointer}_t> \text{ \textit{m}} \_ \textit{proximityResults}
Functionality

- clear()
- preprocess(double ptime)
- solve(double ptime)

```c
void OOPSMP::PRM::preprocess(const double rtime)
{
    OOPSMPClock_t clk;
    double t = 0.0;
    OOPSMPS::set_start_time(&clk);

    m_undirected = getCoreRobotData()->getPathGenerator()->isPathReversible();

    if (m_minNrSamples > 0 && m_roadmap->getNrVertices() < m_minNrSamples)
    {
        printf("time left = %f\n", rtime);
        printf("  current and min nr. samples = %5d %5d\n", m_roadmap->getNrVertices(), m_minNrSamples);
        1. generateManySamples(m_minNrSamples - m_roadmap->getNrVertices(), rtime);
        printf("  current and min nr. samples = %5d %5d\n", m_roadmap->getNrVertices(), m_minNrSamples);
    }

    if ((t = OOPSMPElapsed_time(&clk)) < rtime && m_idConnectSample < m_roadmap->getNrVertices())
    {
        printf("time left = %f\n", rtime - t);
        2. ConnectSamples(rtime - t);
    }

    if ((t = OOPSMPElapsed_time(&clk)) < rtime)
    {
        printf("time left = %f\n", rtime - t);
        printf("  begin generating additional samples and connections\n");
        3. generateAndConnectSamples(rtime - t);
        printf("  end generating additional samples and connections %5d..%5d\n", m_idConnectSample, m_roadmap->getNrVertices());
    }
}
```
Functionality

```c
void OOPSM::PRM::generateManySamples()
{
    CoreRobotData_t crd = getCoreRobotData();
    ValidStateSampler_t vss = crd->get();
    StateSpace_t ss = crd->get();
    State_t x = ss->allocState();
    int i = 0;
    double t = 0.0;
    OOPSMclock_t clk;

    OOPSMset_start_time(&clk);
    while(i < n && (t = OOPSMelapsed_time(&clk)) < rtime)
    {
        if(vss->sample(x, rtime - t))
        {
            addSample(x);
            x = ss->allocState();
            i++;
        }
    }
    ss->freeState(x);
}
```

```c
int OOPSM::PRM::addSample(OOPSM::State_t x)
{
    const int id = m_roadmap->getNrVertices();
    VertexData_t v = new VertexData(id, x);
    m_roadmap->addVertex(id, v);
    m_proximity->addKey(v);
    if(m_undirected)
        m_disjointSet->make(id);
    return id;
}
```

1. `generateManySamples(int n, double rtime)`

- Update the data structures

The code snippet demonstrates the implementation of a function to generate many samples in a planning algorithm, focusing on preprocessing and sample generation.
void OOPSMP::PRM::addEdge(const int idFrom, const int idTo, OOPSMP::Path_t path)
{
    m_roadmap->addEdge(idFrom, idTo, new EdgeData(path));
    if(m_undirected)
    {
        assert(path->isReversible());
        m_roadmap->addEdge(idTo, idFrom, new EdgeData(path->reverse()));
        m_disjointSet->join(idFrom, idTo);
    }
}

void OOPSMP::PRM::connectSample(const int idFrom, const int k)
{
    if(m_proximity->isDataStructureConstructed() == false)
    {
        printf(" begin constructing proximity data structure: \n", m_roadmap->getNrVertices());
        m_proximity->setKeyDistFn(proximityKeyDistFn, getCoreRobotData()->getStateSpace());
        m_proximity->constructDataStructure();
        printf(" end    constructing proximity data structure\n");
    }
    LocalPlanner_t localPlanner = getCoreRobotData()->getLocalPlanner();
    VertexData_t vFrom = dynamic_cast<VertexData_t>(m_roadmap->getVertexData(idFrom));
    Path_t path = NULL;

    m_proximityQuery.setNrNeighbors(k);
    m_proximityQuery.setKey(vFrom);
    m_proximity->knearest(&m_proximityQuery, &m_proximityResults, NULL);
    const int nrResults = m_proximityResults.getNrValidResults(HUGE_VAL);
    for(int i = 0; i < nrResults; i++)
    {
        VertexData_t vTo = (VertexData_t) (m_proximityResults.getKey(i));
        if(shouldAddEdge(idFrom, vTo->m_id) &&
            (path = localPlanner->generateValidForwardConnectPath(vFrom->m_x, vTo->m_x)))
            addEdge(idFrom, vTo->m_id, path);
    }
}

Compute neighbors

If local path is free, then add edge
Functionality

preprocess( double ptime )
  1. generateManySamples( int n, double rtime )
  2. connectSamples( double time )
  3. generateAndConnectSamples( double rtime )

```cpp
void OOPSMP::PRM::generateAndConnectSamples(const double rtime)
{
    OOPSMPclock_t clk;
    int nrVertices = m_roadmap->getNrVertices();

    OOPSMPset_start_time(&clk);
    for(double t = rtime; t > 0.0; t -= OOPSMPelapsed_time(&clk))
    {
        OOPSMPset_start_time(&clk);
        if(m_maxNrSamples < 0 || nrVertices < m_maxNrSamples)
        {
            generateSamples(t * m_fractionSamplingTime);
            nrVertices = m_roadmap->getNrVertices();
        }
        if(m_idConnectSample < nrVertices)
        {
            connectSamples(t - OOPSMPelapsed_time(&clk));
            if(m_maxNrSamples >= 0 && nrVertices >= m_maxNrSamples && m_idConnectSample >= nrVertices)
                break;
        }
    }
}
```
Directions in Roadmap Planning

• Guarantee coverage and connectivity with a small number of nodes
  [Simeon, Laumond, Nissoux ‘00] - [Geraerts, Overmars ‘04] - [Amato et al. ‘06] etc.

Example Case:
• Visibility-based PRM
  – Add samples only if they are:
    • Guards
      (cannot by connected to roadmap nodes)
    • Connectors
      (connect at least two disconnected components of the roadmap)
  – Stop when M attempts to add a node have failed
    • Natural probabilistic criterion (1-1/M prob. of covering the space)
  – Small roadmaps
1. Create a new class that just extends the planner without any functionality

```cpp
#ifndef OOPSMP_MOTION_PLANNERS_PRM_FOO_PRM__OOPSMP_FOO_PRM_H_
#define OOPSMP_MOTION_PLANNERS_PRM_FOO_PRM__OOPSMP_FOO_PRM_H_

#include "MotionPlanners/PRM/OOPSMPPRM.H"

namespace OOPSMP
{
    ForwardClassDeclaration(FooPRM);

    class FooPRM : public PRM
    {
      public:

        FooPRM(void) : PRM()
        {}

        virtual ~FooPRM(void)
        {}

    };

    DeclareInstanceFactory(FooPRM, PRMFactory);
}
#endif
```

2. If you want, create the equivalent graphical version

3. Create an input file that calls the new class

4. Add the new classes in the CMakeLists.txt so that they are compiled

5. Make sure compiles and runs – the old planner should be called and executed properly
First Steps in Extending a Planner

1. Create a new class that just extends the planner without any functionality

```cpp
#include "Graphics/GMotionPlanners/GPRM/GFooPRM/OOPSMPGFooPRM.H"
BeginImplementInstanceFactory(GFooPRM, GPRMFactory);
EndImplementInstanceFactory(GFooPRM);
```

2. If you want, create the equivalent graphical version

```cpp
#ifndef OOPSMP_GRAPHICS_GMotionPlanners_GPRM_GFooPRM_OOPSMP_GFooPRM_H
#define OOPSMP_GRAPHICS_GMotionPlanners_GPRM_GFooPRM_OOPSMP_GFooPRM_H

#include "Graphics/GMotionPlanners/GPRM/OOPSMPGPRM.H"
#include "MotionPlanners/PRM/FooPRM/OOPSMPFooPRM.H"

namespace OOPSMP {

forwardClassDeclaration(GFooPRM);

class GFooPRM : public GPRM {
public:

GFooPRM(void) : GPRM() {}

~GFooPRM(void) {}

};

DeclareInstanceFactory(GFooPRM, GPRMFactory);

#endif
```

3. Create an input file that calls the new class

```cpp
4. Add the new classes in the CMakeLists.txt so that they are compiled

5. Make sure compiles and runs – the old planner should be called and executed properly
1. Create a new class that just extends the planner without any functionality.
2. If you want, create the equivalent graphical version.
3. Create an input file that calls the new class.
4. Add the new classes in the CMakeLists.txt so that they are compiled.
5. Make sure compiles and runs – the old planner should be called and executed properly.

First Steps in Extending a Planner

```xml
<call fn="setMotionPlanner"><arg type="pointer">
  <factory instance="FooPRM"/>
</call>

<call fn="setProximity"><arg type="pointer">
  <factory instance="GNATProximityKeyPointer"/>
</factory>
</call>

<call fn="setNrNeighborsSample"><arg type="int">15</arg></call>
<call fn="setNrNeighborsQuery"><arg type="int">50</arg></call>
<call fn="allowCycles"><arg type="bool">false</arg></call>
<call fn="setFractionSamplingTime">
  <arg type="double">0.2</arg>
</call>
<call fn="setMinMaxNrSamples">
  <arg type="int">2000</arg>
  <arg type="int">5000</arg>
</call>
</factory>
</arg></call>

<call fn="setGMotionPlanner"><arg type="pointer">
  <factory instance="GFooGPRM"/>
</factory>
</call>
```
1. Create a new class that just extends the planner without any functionality
2. If you want, create the equivalent graphical version
3. Create an input file that calls the new class
4. Add the new classes in the CMakeLists.txt so that they are compiled
5. Make sure compiles and runs – the old planner should be called and executed properly
First Steps in Extending a Planner

1. Create a new class that just extends the planner without any functionality

2. If you want, create the equivalent graphical version

3. Create an input file that calls the new class

4. Add the new classes in the CMakeLists.txt so that they are compiled

5. Make sure OOPSMP compiles and runs the new algorithms: the old planner should be called and executed properly
Visibility PRM Functionality

[Visibility PRM Functionality]

Algorithm

\[ ntry \leftarrow 0 \]

\textbf{While} \( ntry < M \)

Select a random free configuration \( q \)

\( \text{guard} \leftarrow \emptyset; G_{vis} \leftarrow \emptyset \)

\textbf{For all components} \( G_i \) of \( \text{Guard} \) \textbf{do}

\( \text{found} \leftarrow \text{FALSE} \)

\textbf{For all nodes} \( g \) of \( G_i \) \textbf{do}

\( \text{If (} g \text{ belongs to } Vis(g) \text{) then} \)

\( \text{found} \leftarrow \text{TRUE} \)

\textbf{If (} \neg \text{guard} \text{) then} \( \text{guard} \leftarrow g; G_{vis} \leftarrow G_i \)

\textbf{Else} /* \( q \) is a connection node */

\( \text{Add } q \text{ to } \text{Connection} \)

\( \text{Create edges } (q, g) \text{ and } (q, \text{guard}) \)

\( \text{Merge components } G_{vis} \text{ and } G_i; \)

\textbf{until} \( \text{found} = \text{TRUE} \)

\( \text{If (} \neg \text{guard) then} /* \( q \) is a guard node */ \)

\( \text{Add } \{g\} \text{ to } \text{Guard}; ntry \leftarrow 0 \)

\textbf{Else} \( ntry \leftarrow ntry + 1 \)

\textbf{End}

[Simeon, Laumond, Nissoux ‘00]
class GuardVisibility
{
friend class VisPRM;
friend class SPR;

protected:
    int nr_connections;  // the number of roadmap nodes from different connected components
    int* connections;   // that the current sample is visible from
    Path_t* paths;      // the indices of the nodes from different connected components that can
                       // view the state
    int allocated;      // the paths that connect the state to roadmap nodes which can "see" the state
                       // number of allocated connections and paths

public:
    GuardVisibility()
    {} 

    ~GuardVisibility()
    {} 

    void clearGuardVisibility()
    {} 

    void addConnection( int index, Path_t path )
    {} 

    // is the current state a guard
    bool isGuard() { return (nr_connections == 0); } 

    // is the current state a connector
    bool isConnector() { return (nr_connections >= 2); } 

};

typedef GuardVisibility* GuardVisibility_t;
// is the sampled state already visible from a roadmap on the same // component as the the node: "node_index"?
bool OOPSMP::VisPRM::isNodeVisibleFromComponent( int node_index )
{
    // is the state already visible by a node in the same component as the i-th node?
    bool not_visible_from_component = true;

    for( int i=0; not_visible_from_component == true && i<g_vis->nr_connections; i++ )
        if( m_disjointSet->same( g_vis->connections[i].node_index ) == true )
            not_visible_from_component = false;

    return not_visible_from_component;
}

int nrVertices = m_roadmap->getNrVertices();

// a temporary variable to remember paths
Path_t path = NULL;

// iterate over all nodes
for( int i=0; i<nrVertices; i++ )
{
    State_t vTo = (State_t)( (VertexData_t) m_roadmap->getVertexData(i) )->m_x;

    // is the state already visible by a node in the same component as the i-th node?
    bool notVisibleFromComponent = isNodeVisibleFromComponent( i );

    // consider the i-th roadmap node only if the state is not already // visible by a node in the same component like the i-th node
    if( notVisibleFromComponent == true )
    {
        // check whether the i-th node is able to "view" the state
        if( (path = lp->generateValidForwardConnectPath( state, vTo ) ) )
            g_vis->addConnection( i, path );
    }
}
CoreRobotData_t crd = getCoreRobotData();
ss = crd->getStateSpace(); // State Space
vss = crd->getValidStateSampler(); // Valid State Sampler
lp = getCoreRobotData()->getLocalPlanner(); // Local Planner

// initial number of efforts to add a guard/roadmap node is 0
int nr_tries = 0;
g_vis = new GuardVisibility();

// keep trying while we have not exceeded the
// maximum number of failed attempts to add a guard
while( nr_tries < max_tries )
{
    // sample a random valid state
    1. sampleState();

    // compute the nodes from different connected components that can view the sample
    2. computeGuardVisibility();

    if( g_vis->isGuard() || g_vis->isConnector() ) // this is a guard or a "connection" node
        nr_tries = 0;
    3a. addGuard();
    else
        3b. ss->freeState( state );
          nr_tries++;

    g_vis->clearGuardVisibility();
}

// add the sampled state as a guard
delete( g_vis );
void OOPSM::VisPRM::addGuard()
{
    int node_index = m_roadmap->getNrVertices();
    addSample( state );
    for( int i=0; i<g_vis->nr_connections; i++ )
        addEdge( node_index, g_vis->connections[i], g_vis->paths[i] );
}
Example
Developing New Planners

• Quality of paths returned by previous techniques is low
  – Can we improve path quality returned directly by the planner
    • Before smoothing
  – Retain connectivity and coverage properties of VisPRM

[Jailllet, Simeon '06] – [Geraerts, Overmars '05-'07] etc.

Useful Edge Criterion [Nieuwenhuisen, Overmars '04]

Is edge \((q,q')\) useful?

Yes, if: \(K \cdot d(q,q') < G(q,q')\)

\(d(q,q')\): C-space distance
\(G(q,q')\): graph distance (\(\infty\) if no path)
A sub-graph \( G' \) is a spanner of a graph \( G \) if:

\[
\pi_{G'}(p,q) \leq s \cdot \pi_G(p,q)
\]

for constant \( s \) and for all pairs of nodes \( p \) and \( q \) in \( G \), where \( \pi_G(p,q) \) denotes the shortest path distance between \( p \) and \( q \) in the graph \( G \).

**Direction:**
Implicitly and incrementally build a spanner of the full graph defined by the sampled configurations

(Based on incremental spanner graph construction by [Baswana, Sen ’08])
(Read more about incremental spanners: [Gao, Guibas, Nguyen ‘04])
Spanner Roadmaps

Objective: Small roadmaps with guarantees about path quality (bounded by K*optimal)

- We store the failed samples
- Defines a bipartite graph
- The roadmap is a skeleton (spanner graph) of the complete graph

- Nodes have visibility limits
- Nodes are still added as guards and connectors
- Cycles are allowed and added between roadmap nodes
- Secondary nodes can later be inserted into the roadmap by defining a useful cycle
// the class to store the information of samples that are not added
directly to the roadmap
// these states may be added later on to the roadmap

class SecondaryNode{
  
friend class SPR;

protected:
  State_t state;
  int guardIndex;
  double distance;
  bool isAdded;

public:
  SecondaryNode( State_t st = NULL, int gIn = -1, double dis = 0 )
  { setInfo( st, gIn, dis ); }
  virtual ~SecondaryNode() {}

  State_t getState() { return state; }
  int getGuard() { return guardIndex; }
  double getDistance() { return distance; }
  bool hasBeenAdded() {return isAdded; }

  void setInfo( State_t st, int gIn, double dis )
  { state = st; guardIndex = gIn; distance = dis; isAdded = false; }
  void setNewGuard( int gIn, double dis )
  { guardIndex = gIn; distance = dis; }

  void addAsGuard()
  { isAdded = true; }
};
typedef SecondaryNode* SecondaryNode_t;
// initial number of efforts to add a guard/roadmap node is 0
int nr_tries = 0;

allocateProximityStructures();

g_vis = new GuardVisibility();

// keep trying while we have not exceed the
// maximum number of failed attempts to add a guard
while( nr_tries < max_tries )
{
    // sample a random valid state
    sampleState();

    // compute the nodes from different connected components that can view the sample
    computeGuardVisibility();

    bool added = true;
    if( g_vis->isGuard() || g_vis->isConnector() ) // this is a guard or a "connection" node
        addGuard();
    else
        added = false;

    if( added )
    {
        int latest_sample_index = m_roadmap->getNrVertices() - 1;

        // connect to other roadmap nodes to create useful cycles
        addUsefulCycles( latest_sample_index );

        // change the guard info of the closest secondary nodes
        updateGuards( stateNode, latest_sample_index );
    }

    // check neighboring upgrade nodes for upgrades
    added = checkForUpgrades( stateNode, added );

    if( !added )
    {
        addSecondaryNode( stateNode );
        nr_tries++;
        // another failed attempt to add a new node
    }
    else
    {
        nr_tries = 0;
        delete( stateNode );
    }
4. Adding Useful Cycles

```cpp
void OOPSMP::SPR::addUsefulCycles( int nodeIndex )
{
    Path_t path;

    // the state at the nodeIndex
    VertexData_t vFrom = dynamic_cast<VertexData_t>(m_roadmap->getVertexData(nodeIndex));
    State_t sFrom = (State_t)vFrom->m_x;

    // find the 20 closest neighbors to nodeIndex
    m_proximityQuery.setNrNeighbors( 10 );
    m_proximityQuery.setKey( vFrom );
    m_proximity->knearest( &m_proximityQuery, &m_proximityResults, NULL );
    const int nrResults = m_proximityResults.getNrValidResults(HUGE_VAL);

    // iterate over the neighbors as long as their distance is less than distVis
    // attempt to add an edge if we have a useful cycle: K * state_dis < graph_dis
double stateDis = 0, graphDis;
    for( int i = 0; i < nrResults && stateDis < distVis; i++ )
    {
        VertexData_t vTo = (VertexData_t)(m_proximityResults.getKey(i));
        State_t sTo = (State_t)vTo->getState();
        int toIndex = vTo->getId();

        graphDis = graphDistance( nodeIndex, toIndex );
        stateDis = ss->distanceStates( sFrom, sTo );

        if( K * stateDis < graphDis )
            if( (path = lp->generateValidForwardConnectPath( sFrom, sTo )) )
                addEdge( nodeIndex, toIndex, path );
    }
}
```
5/6: Dealing with Secondary Nodes

```cpp
void OOPSMP::SPR::allocateProximityStructures()
{
    if( m_proximity->isDataStructureConstructed() == false )
    {
        m_proximity->setKeyDistFn( proximityKeyDistFn, getCoreRobotData()->getStateSpace() );
        m_proximity->constructDataStructure();
    }

    if( sec_proximity->isDataStructureConstructed() == false )
    {
        sec_proximity->setKeyDistFn( secProximityKeyDistFn, getCoreRobotData()->getStateSpace() );
        sec_proximity->constructDataStructure();
    }
}

void OOPSMP::SPR::updateGuards( SecondaryNode_t stateNode, int nodeIndex )
{
    Path_t path;
    State_t staState = (State_t) stateNode->getState();

    // find the 100 secondary nodes closest to nodeIndex
    sec_proximityQuery.setNrNeighbors( 50 );
    sec_proximityQuery.setKey( stateNode );
    sec_proximity->knearest( &sec_proximityQuery, &sec_proximityResults, NULL );
    const int nrResults = sec_proximityResults.getNrValidResults(HUGE_VAL);

    // iterate over the neighbors as long as their distance is less than distVis
    double stateDis = 0;
    for( int i=0; i<nrResults && stateDis < distVis; i++ )
    {
        SecondaryNode_t secNode = (SecondaryNode_t) (sec_proximityResults.getKey(i));

        State_t secState = (State_t) secNode->getState();
        stateDis = ss->distanceStates( staState, secState );

        if( secNode->hasBeenAdded() == false )
            if( stateDis < secNode->getDistance() )
                if( (path = lp->generateValidForwardConnectPath( staState, secState )) )
                    secNode->setNewGuard( nodeIndex, stateDis );
    }
}
Example
Example
Example
Summary

• Using OOPSMP we showed that
  – Easy to create new roadmap planners
  – We can create more complicated but powerful algorithms given the basic planning modules in OOPSMP

• OOPSMP + ScetchUp allow
  – Common benchmarks and an easy way to construct new ones
  – An easy way to compare between alternative techniques using common tools (e.g., collision detection, state representation, local planner)
  – Simplify the process of designing new algorithms
  – And streamline the experimental process