Optimization methods for the memory allocation problems in embedded systems

María Soto

Advisor: Marc Sevaux
Co-advisor: André Rossi

Lab-STICC
Université de Bretagne-Sud
Lorient - FRANCE
maria.soto@univ-ubs.fr

September 29, 2011

Outline

1. Introduction
   - Embedded systems
   - Memory allocation
   - Conflict graph
2. Memory allocation problems
   - Unconstrained memory allocation problem
   - Allocation with constraint on the number of memory banks
   - General memory allocation problem
   - Dynamic memory allocation problem
3. Conclusions and future work

Outline

1. Introduction
   - Embedded systems
   - Memory allocation
   - Conflict graph
2. Memory allocation problems
3. Conclusions and future work

What is an embedded system?

Embedded system

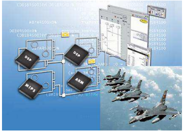
Minicomputer designed to satisfy specific requirements

Embedded Systems

Daily activities

Embedded systems

Industry



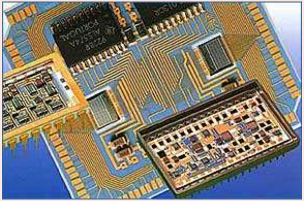
Design challenge

Technology offers more and more functionalities

Design challenge

Technology offers more and more functionalities

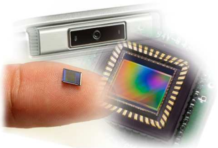
Design of embedded systems becomes more and more complex



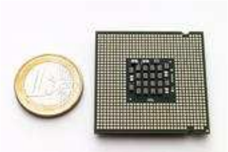
Designer objectives

Designers must take into account:

Power consumption (W)



Area (mm^2)



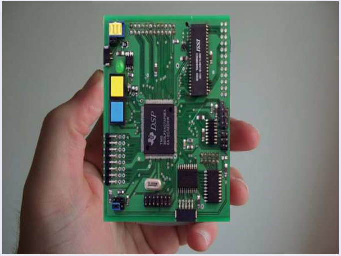
Cost ($)

Architecture

Embedded system

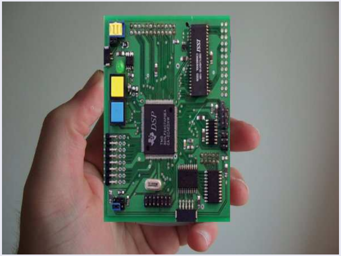
Architecture

Embedded system

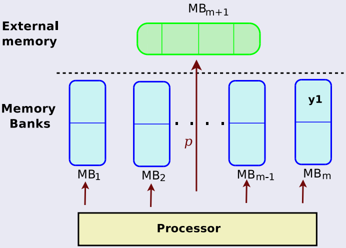


Architecture

Embedded system



Simplified architecture



Application

Embedded system

Functionalities

- phoning
- texting
- digital camera
- web browsing

etc...

Application

Embedded system

Functionalities

- phoning
- texting
- digital camera
- web browsing

etc...

C source code

```
/* LMS dual-channel filter */

void main()  {
  int y11, y12, y21, y22, e1, e2;
  for(int k=0;k<10;k++)
  {
    int n = (k+10)%L;
    y11=0;
    for(int i=0;i<10;i++)
       {   y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
    y12=0;
    for(int i=0;i<10;i++)
       {   y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
    e1 = y1[n]-y11-y12;  /* error */
    H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
    H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
    y21=0;
    for(int i=0;i<10;i++)
       {   y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
    y22=0;
    for(int i=0;i<10;i++)
       {   y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
     e2 = y2[n]-y21-y22;
     H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
     H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Data structures

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
    int y11, y12, y21, y22, e1, e2;
    for(int k=0;k<10;k++)
    {
      int n = (k+10)%L;
      y11=0;
      for(int i=0;i<10;i++)
        {   y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
      y12=0;
      for(int i=0;i<10;i++)
        {   y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
      e1 = y1[n]-y11-y12;  /* error */
      H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
      H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
      y21=0;
      for(int i=0;i<10;i++)
        {   y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
      y22=0;
      for(int i=0;i<10;i++)
        {   y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
       e2 = y2[n]-y21-y22;
       H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
       H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Data structures

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {   y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {   y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {   y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {   y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

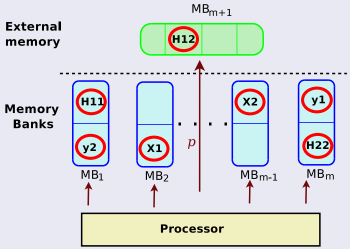
Data structures

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Memory architecture



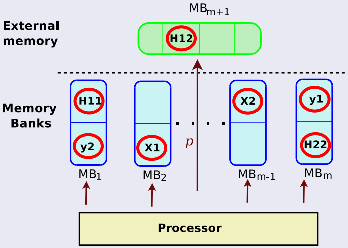
Data structures

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Memory architecture



Memory allocation impacts power consumption, area and cost

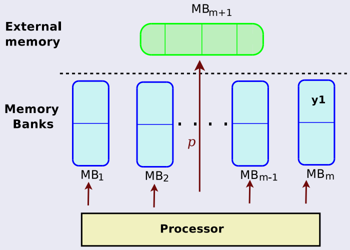
Processor

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Memory architecture



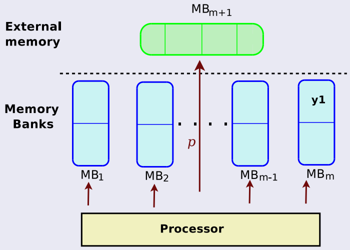
Processor

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
    int y11, y12, y21, y22, e1, e2;
    for(int k=0;k<10;k++)
    {
      int n = (k+10)%L;
      y11=0;
      for(int i=0;i<10;i++)
          {  y11 =    X1   *   H11         y11; }
      y12=0;
      for(int i=0;i<10;i++)
          {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
      e1 = y1[n]-y11-y12;  /* error */
      H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
      H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
      y21=0;
      for(int i=0;i<10;i++)
          {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
      y22=0;
      for(int i=0;i<10;i++)
          {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
       e2 = y2[n]-y21-y22;
       H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
       H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Memory architecture



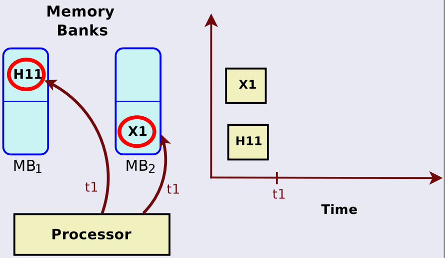
Processor

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 =   X1  *  H11   y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Loading process



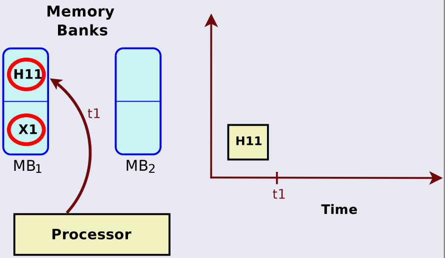
Processor

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = [ X1  *  H11 ] y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Loading process



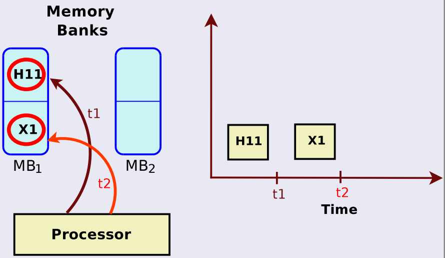
Processor

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
    int y11, y12, y21, y22, e1, e2;
    for(int k=0;k<10;k++)
    {
      int n = (k+10)%L;
      y11=0;
      for(int i=0;i<10;i++)
         {  y11 =     X1  *  H11     y11; }
      y12=0;
      for(int i=0;i<10;i++)
         {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
      e1 = y1[n]-y11-y12;  /* error */
      H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
      H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
      y21=0;
      for(int i=0;i<10;i++)
         {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
      y22=0;
      for(int i=0;i<10;i++)
         {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
       e2 = y2[n]-y21-y22;
       H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
       H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Loading process



Conflicts

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 =       X1   *   H11        y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Conflict

- Two structures are said to be conflicting if they are required at the same time (*e.g.* `X1*H11`)

Conflicts

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+     X1  *  H11     y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Cost conflict

- Conflicts have a cost in milliseconds (ms) (*e.g.* **`X1*H11`** $\rightarrow$ cost of 100 ms)
- Automatically computed by *SoftExplorer* `http://www.softexplorer.fr`

Conflicts

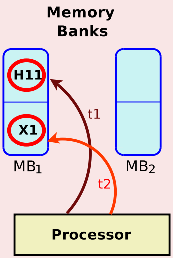
Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+   X1  *  H11   y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Open conflict

X1 and **H11** are allocated in the same memory bank → cost is paid



Conflicts

Application

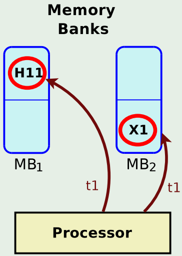
Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+    X1  *  H11    y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Closed conflict

X1 and **H11** are assigned to different memory banks

$\rightarrow$ no cost



Conflicts

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
     e2 = y2[n]-y21-y22;
     H21 [n+1]  =   H21 [n] mu21*X1[n]*e2;
     H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Special cases

- *Auto-conflict*: a data structure is in conflict with itself (always open)
  For example **`H21[n+1] = H21[n]`**

- Isolated data structure: a data structure is not in conflict with any other one

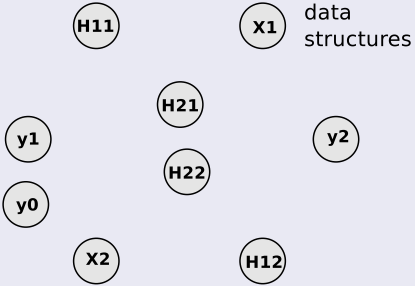
Conflict Graph $G = (X, U)$

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Conflict graph



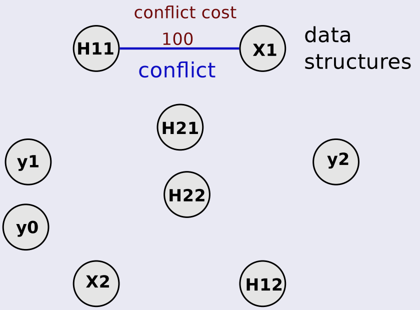
Conflict Graph $G = (X, U)$

Application

Source code

```
/* LMS dual-channel filter */

void main()  {
   int y11, y12, y21, y22, e1, e2;
   for(int k=0;k<10;k++)
   {
     int n = (k+10)%L;
     y11=0;
     for(int i=0;i<10;i++)
        {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
     y12=0;
     for(int i=0;i<10;i++)
        {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
     e1 = y1[n]-y11-y12;  /* error */
     H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
     H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
     y21=0;
     for(int i=0;i<10;i++)
        {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
     y22=0;
     for(int i=0;i<10;i++)
        {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Conflict graph



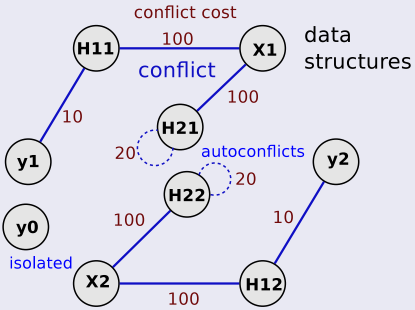
Conflict Graph $G = (X, U)$

Application

Source code

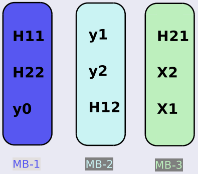
```
/* LMS dual-channel filter */

void main()  {
    int y11, y12, y21, y22, e1, e2;
    for(int k=0;k<10;k++)
    {
      int n = (k+10)%L;
      y11=0;
      for(int i=0;i<10;i++)
         {  y11 = X1[(i+k)%L]*H11[(L-1+k-i)%L]+y11; }
      y12=0;
      for(int i=0;i<10;i++)
         {  y12 = X2[(i+k)%L]*H12[(L-1+k-i)%L]+y12; }
      e1 = y1[n]-y11-y12; /* error */
      H11[(n+1)%L] = H11[n]+mu11*X1[n]*e1;
      H12[(n+1)%L] = H12[n]+mu12*X2[n]*e1;
      y21=0;
      for(int i=0;i<10;i++)
         {  y21 = X1[(i+k)%L]*H21[(L-1+k-i)%L]+y21; }
      y22=0;
      for(int i=0;i<10;i++)
         {  y22 = X2[(i+k)%L]*H22[(L-1+k-i)%L]+y22; }
      e2 = y2[n]-y21-y22;
      H21[(n+1)%L] = H21[n]+mu21*X1[n]*e2;
      H22[(n+1)%L] = H22[n]+mu22*X2[n]*e2; }  }
```

Conflict graph



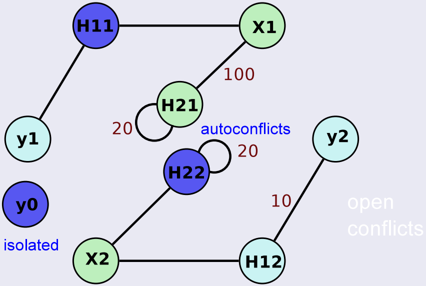
Solution

Memory allocation



Memory banks ⟶ colors

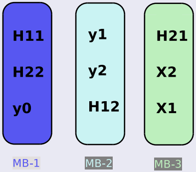
Coloration



Total cost: 150 ms

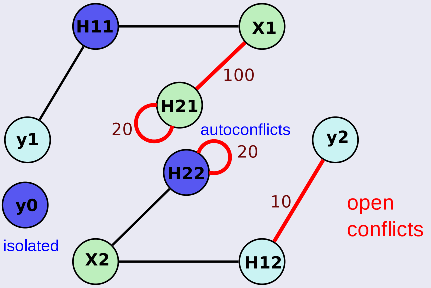
Solution

Memory allocation



Memory banks ⟶ colors

Coloration



Total cost: 150 ms

Outline

1. Introduction

2. Memory allocation problems
   - Unconstrained memory allocation problem
   - Allocation with constraint on the number of memory banks
   - General memory allocation problem
   - Dynamic memory allocation problem

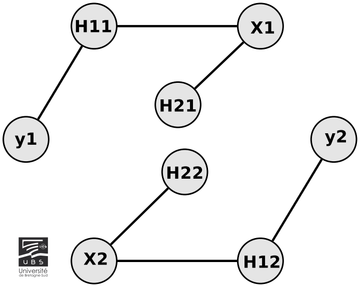
3. Conclusions and future work

Unconstrained memory allocation problem

- From now, auto-conflicts are ignored because they are always open

Objective

Find the minimum number of memory banks for which all conflicts are closed



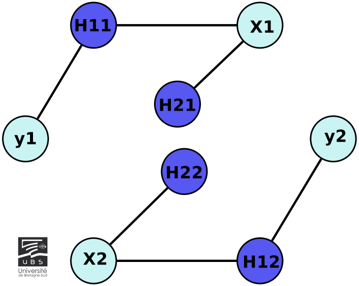
Finding the minimum number of colors to color graph
$\rightarrow$ **Chromatic number** χ

Unconstrained memory allocation problem

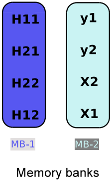
- From now, auto-conflicts are ignored because they are always open

Objective

Find the minimum number of memory banks for which all conflicts are closed



2 colors $\rightarrow$ 2 memory banks



New upper bounds

- Easily computable, even for large graphs
- Outperform the upper bounds from literature
- But far from the optimal solution

- Sophisticated methods spend too much time
- Problem solved repeatedly in CAD software

- Upper bounds provide a satisfactory solution

Allocation with constraint on the number of memory banks

- Fixed number of memory banks
- Cost of conflicts

Objective

Find memory allocation for data structures such that total cost of open conflicts is minimized

- k-weighted graph coloring problem

Proposed approach from operations research:
$\rightarrow$ Memetic Algorithm hybridized with Tabu Search

Allocation with constraint on the number of memory banks

- Fixed number of memory banks
- Cost of conflicts

Objective

Find memory allocation for data structures such that total cost of open conflicts is minimized

- k-weighted graph coloring problem

Proposed approach from operations research:
$\rightarrow$ Memetic Algorithm hybridized with Tabu Search

Allocation with constraint on the number of memory banks

- Fixed number of memory banks
- Cost of conflicts

Objective

Find memory allocation for data structures such that total cost of open conflicts is minimized

- k-weighted graph coloring problem

Proposed approach from operations research:
$\rightarrow$ Memetic Algorithm hybridized with Tabu Search

UBS Université de Bretagne-Sud

Lab-STICC

Memetic Algorithm

Population

Solutions

Memetic Algorithm

Population

Solutions

Random selection

Cross parents

Parents solutions

Memetic Algorithm

Population

Solutions

Random selection

Cross parents

Parents solutions

Better characteristics

Offspring

New solution

Memetic Algorithm

Population

Solutions

Random selection

Cross parents

Parents solutions

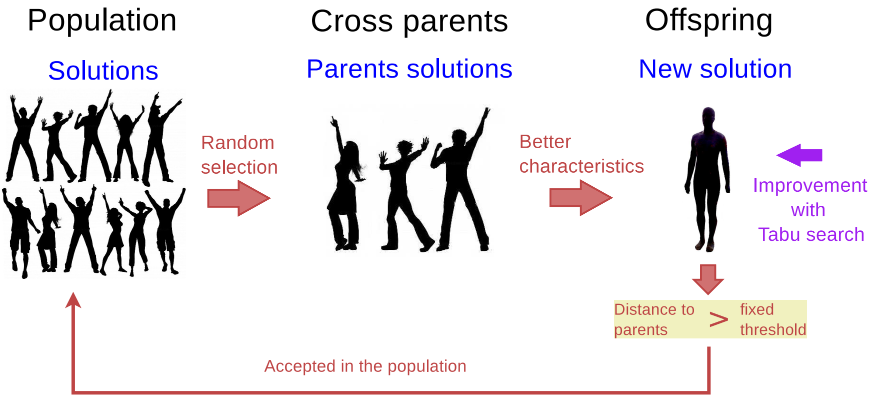
Better characteristics

Offspring

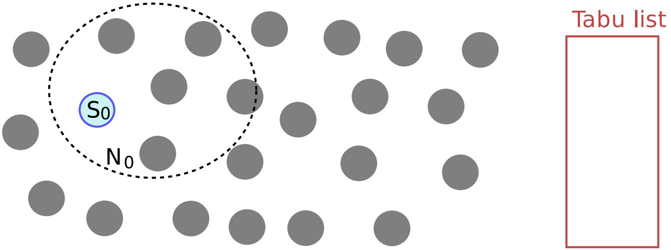
New solution

Improvement with Tabu search

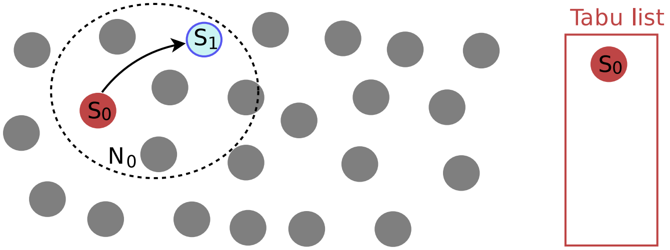
Memetic Algorithm



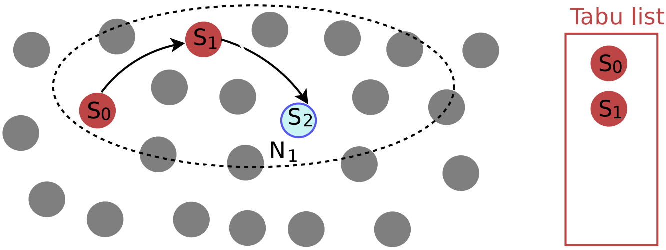
Tabu Search



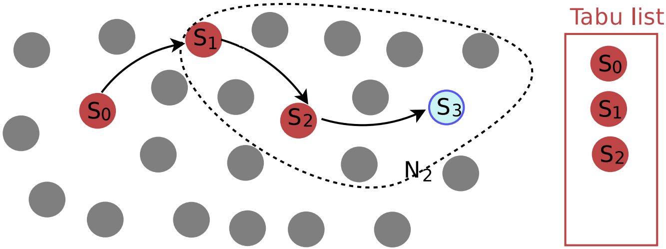
Tabu Search



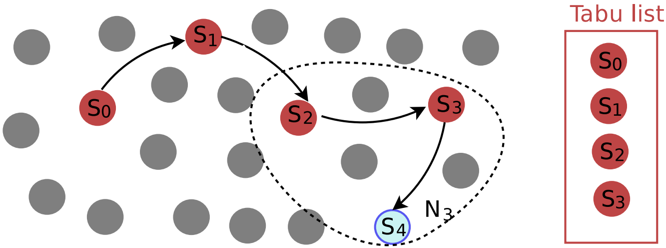
Tabu Search



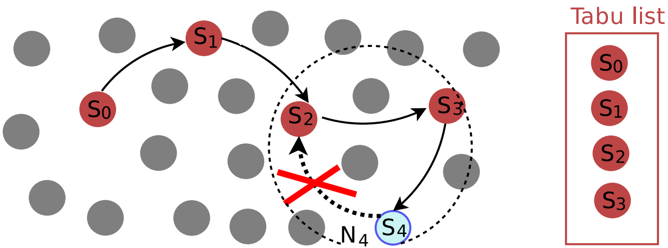
Tabu Search



Tabu Search



Tabu Search



General memory allocation problem

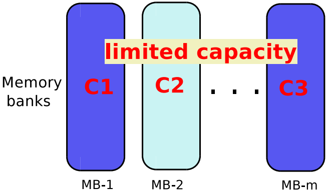
Same characteristics as previous problem, plus

- Sizes of data structures
- Number of accesses of data structures

General memory allocation problem

Same characteristics as previous problem, plus

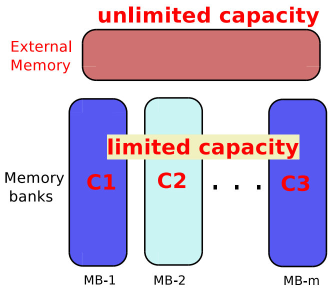
- Sizes of data structures
- Number of accesses of data structures
- Limited capacity of memory banks



General memory allocation problem

Same characteristics as previous problem, plus

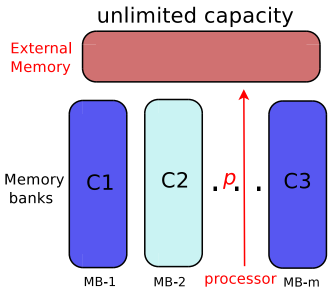
- Sizes of data structures
- Number of accesses of data structures
- Limited capacity of memory banks
- External memory $\rightarrow$ unlimited capacity



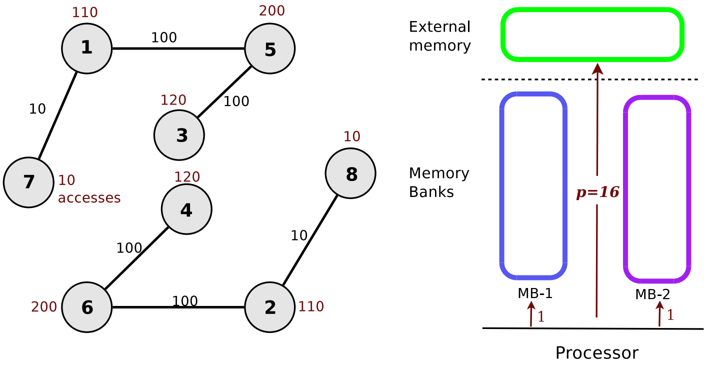
General memory allocation problem

Same characteristics as previous problem, plus

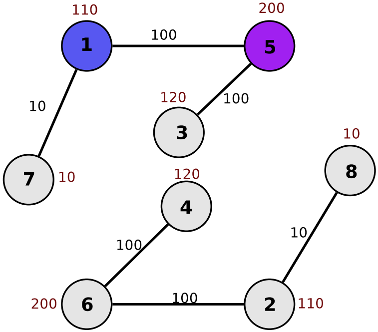
- Sizes of data structures
- Number of accesses of data structures
- Limited capacity of memory banks
- External memory → unlimited capacity
- Access to external memory → p

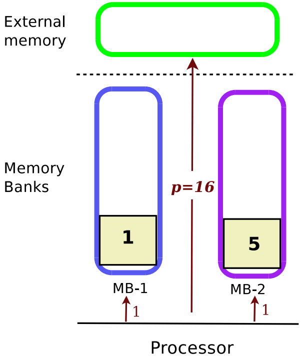


Example

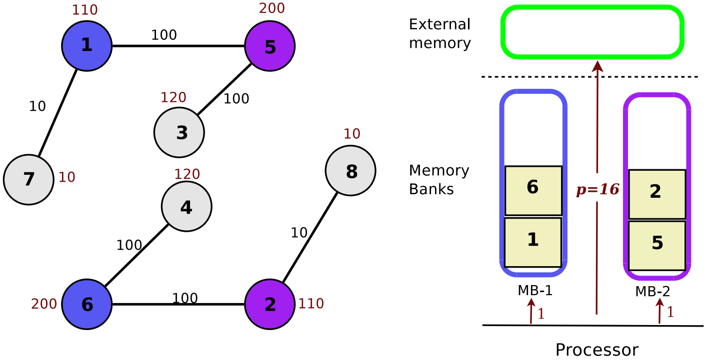

110
200
1
100
5
10
120
100
3
10
8
7
10
accesses
120
4
100
10
200
6
100
2
110
External
memory
Memory
Banks
p=16
MB-1
1
MB-2
1
Processor

Example

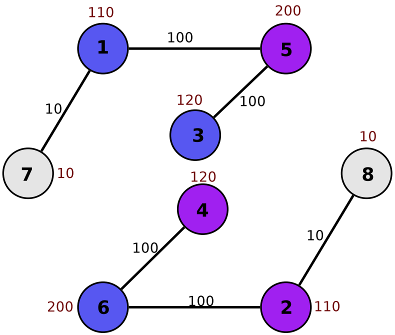

110
1
100
200
5
10
120
100
3
10
8
7
10
120
4
100
10
200
6
100
2
110

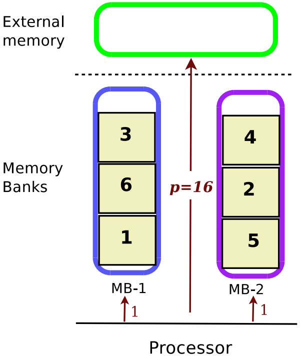

External memory
Memory Banks
p=16
1
5
MB-1
MB-2
1
1
Processor

Example

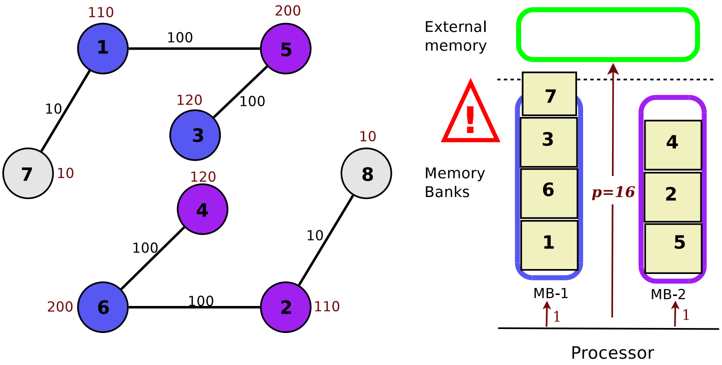

110
200
100
1
5
10
120
100
3
10
8
7
10
120
4
100
10
200
6
100
2
110
External memory
Memory Banks
6
1
p=16
2
5
MB-1
MB-2
1
1
Processor

Example

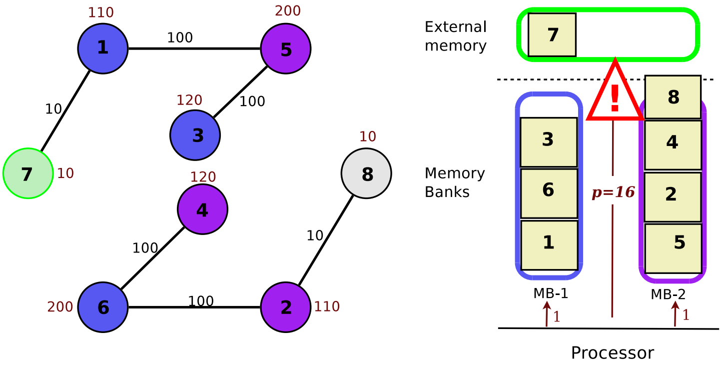

110
1
100
200
5
10
120
100
3
7
10
10
8
120
4
100
10
200
6
100
2
110


External memory
Memory Banks
3
6
1
p=16
4
2
5
MB-1
MB-2
1
1
Processor

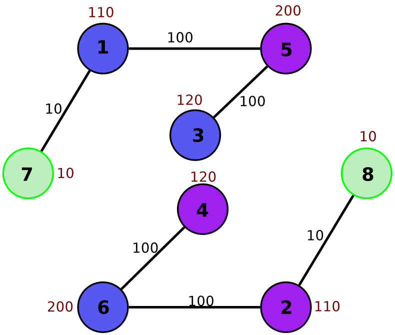
Example

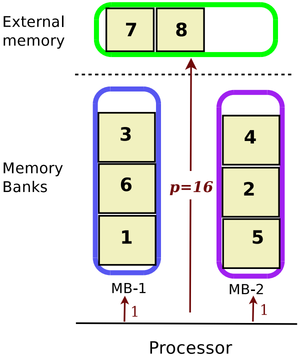

110
200
1
100
5
10
120
100
3
7
10
10
8
120
4
100
10
200
6
100
2
110
External memory
Memory Banks
7
3
6
1
4
2
5
p=16
MB-1
MB-2
1
1
Processor

Example


110
200
100
1
5
10
120
100
3
10
7
10
120
8
4
100
10
200
6
100
2
110
External memory
7
8
3
4
6
p=16
2
Memory Banks
1
5
MB-1
MB-2
1
1
Processor

Example


110
200
1
100
5
10
120
100
3
10
7
10
8
120
4
100
10
200
6
100
2
110


External memory
7
8
Memory Banks
3
6
1
p=16
4
2
5
MB-1
MB-2
1
1
Processor

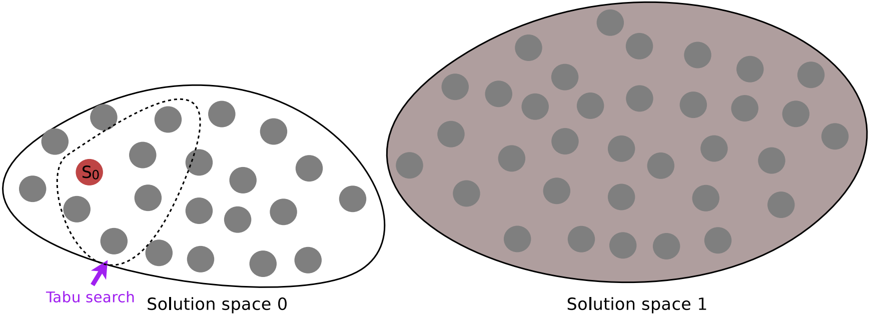
General memory allocation problem

Objective:

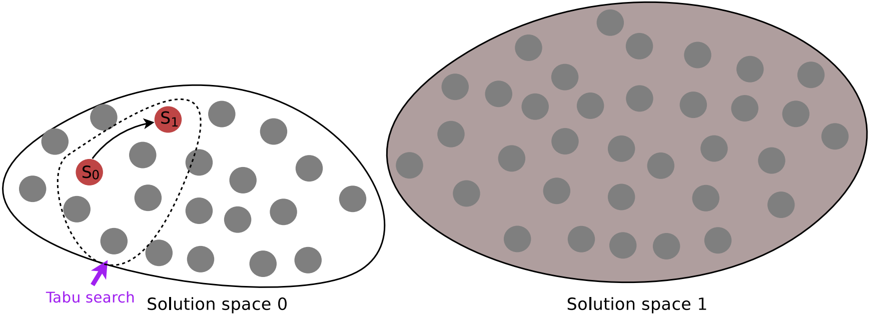
For a given number of capacitated memory banks and an external memory, find a memory allocation for data structures such that the time spent accessing these data is minimized.

Proposed approach:
$\rightarrow$ Variable Neighborhood Search-Tabu Search (VNS-TS)

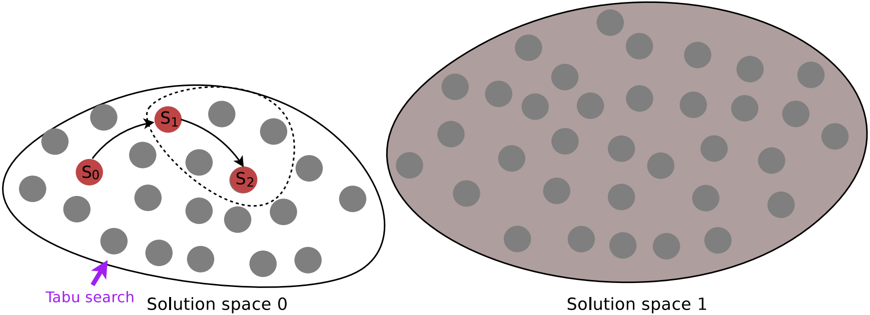
Hybrid approach (VNS-TS)



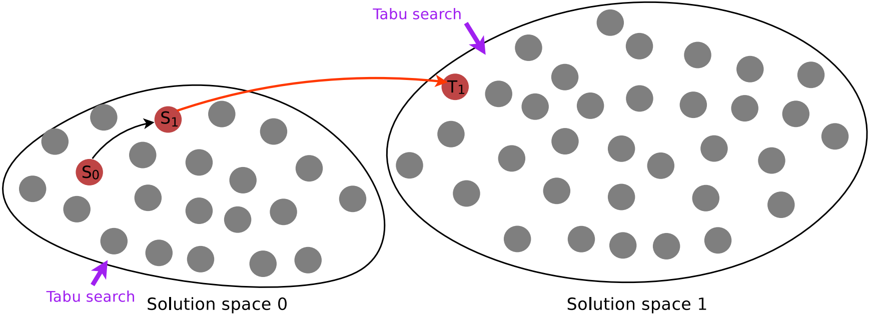
Hybrid approach (VNS-TS)



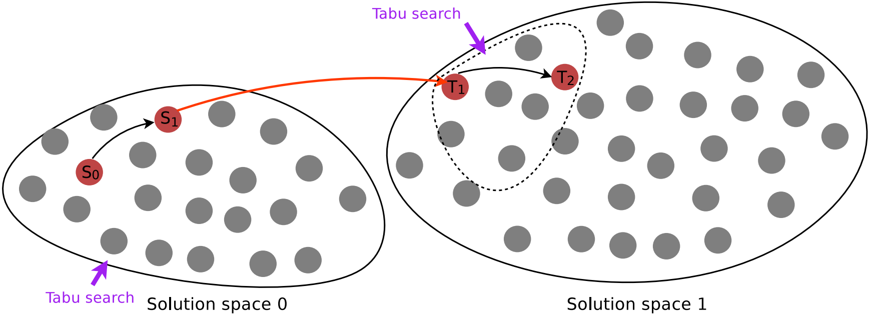
Hybrid approach (VNS-TS)



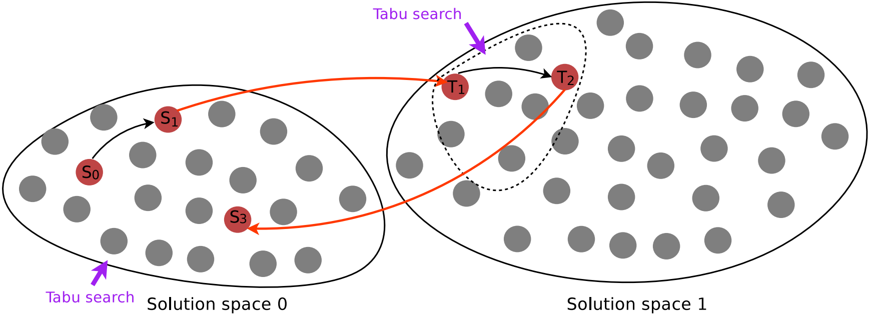
Hybrid approach (VNS-TS)



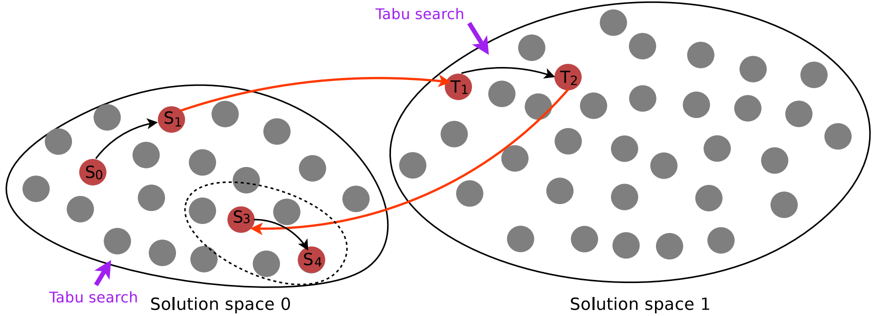
Hybrid approach (VNS-TS)



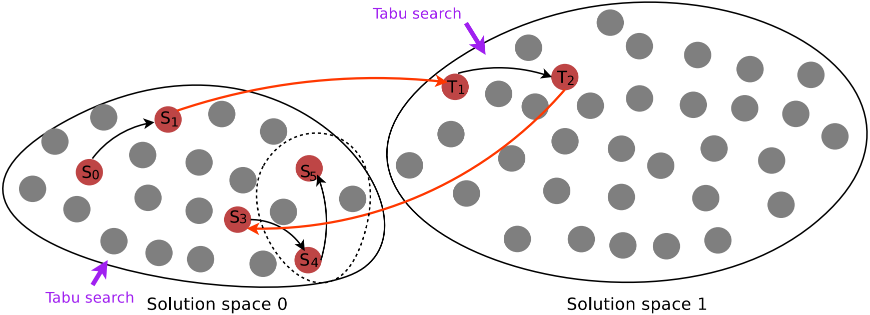
Hybrid approach (VNS-TS)



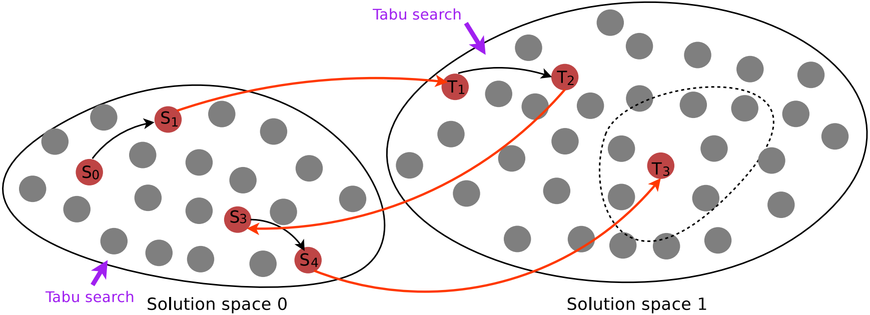
Hybrid approach (VNS-TS)



Hybrid approach (VNS-TS)



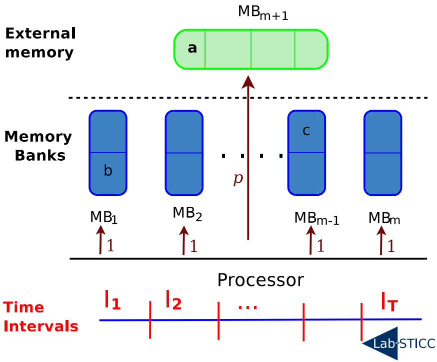
Hybrid approach (VNS-TS)



Dynamic memory allocation problem

Same characteristics as general memory allocation problem, plus

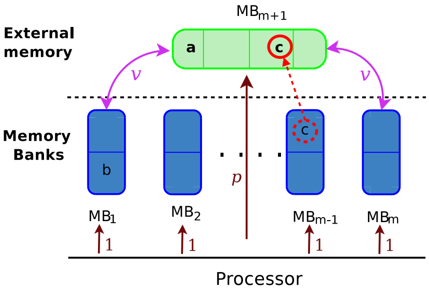
- Application time split into time intervals



Dynamic memory allocation problem

Same characteristics as general memory allocation problem, plus

- Application time split into time intervals
- Transfer rates for data structures
  - from external memory to memory banks, v



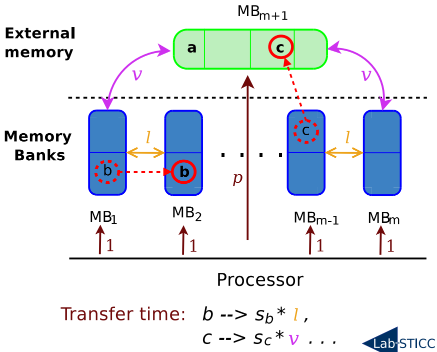
Transfer time:

c --> $s_c * v$ . . .

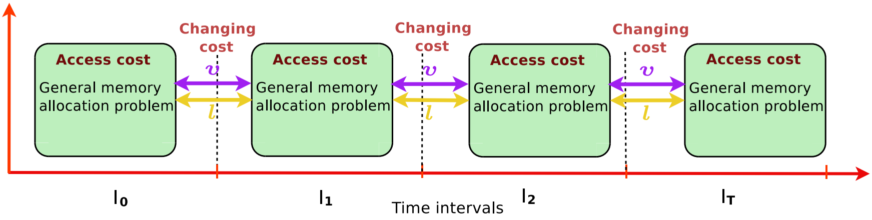
Dynamic memory allocation problem

Same characteristics as general memory allocation problem, plus

- Application time split into time intervals
- Transfer rates for data structures
  - from external memory to memory banks, v
  - between memory banks, l



Dynamic memory allocation problem

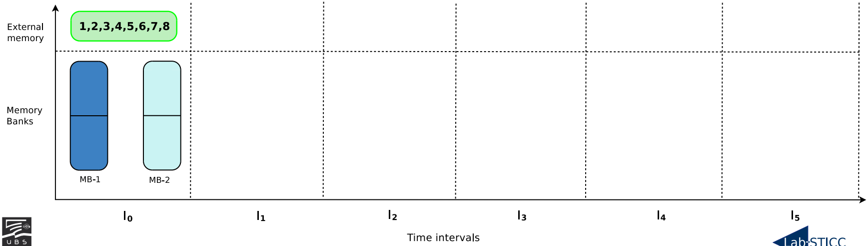


Example

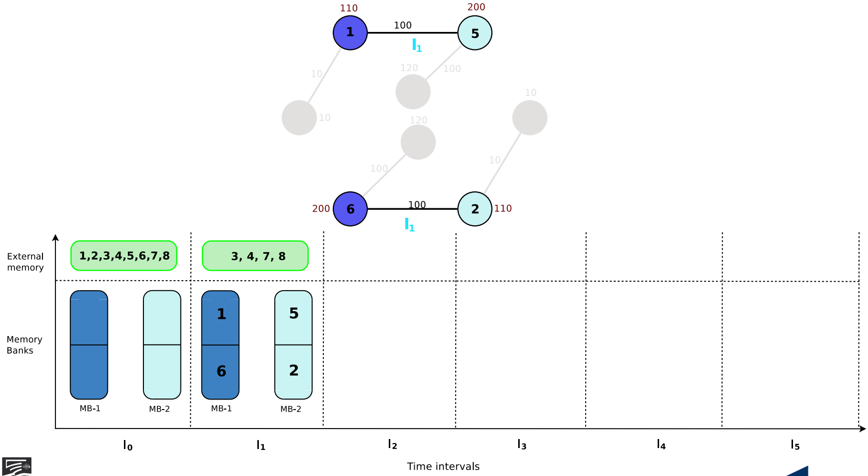
$p = 16$ ms, and $l = v = 1$ ms/ko.

Intervals $t = 1, \ldots, 5$	Data structures $\{a_{1,t}, \ldots, a_{n_t,t}\}$	Conflicts $(a_{k_1,t}, a_{k_2,t})$	Cost $d_{k,t}$	Access time $e_{a_{i,t},t}$
1	{ 1, 5, 2, 6 }	(1;5) (2;6)	1,046,529 1,046,529	$e_{1,1} = e_{2,1} =$ $e_{5,1} = e_{6,1} =$ 1,046,529
2	{ 3, 4, 5, 6 }	(3;5) (4;6)	1,046,529 1,046,529	$e_{3,2} = e_{5,2} =$ $e_{4,2} = e_{6,2} =$1,046,529
3	{ 1,5,7}	(1;7) (1;5)	1,023 1,023	$e_{1,3} =$2,046 $e_{5,3} = e_{7,3} =$1,023
4	{ 2,6,8 }	(2;6) (2;8)	1,023 1,023	$e_{2,4} =$2,046 $e_{6,4} = e_{8,4} =$1,023
5	{ 3,4 }	(3;3) (4;4)	2,046 2,046	$e_{3,5} = e_{4,5} =$2,046

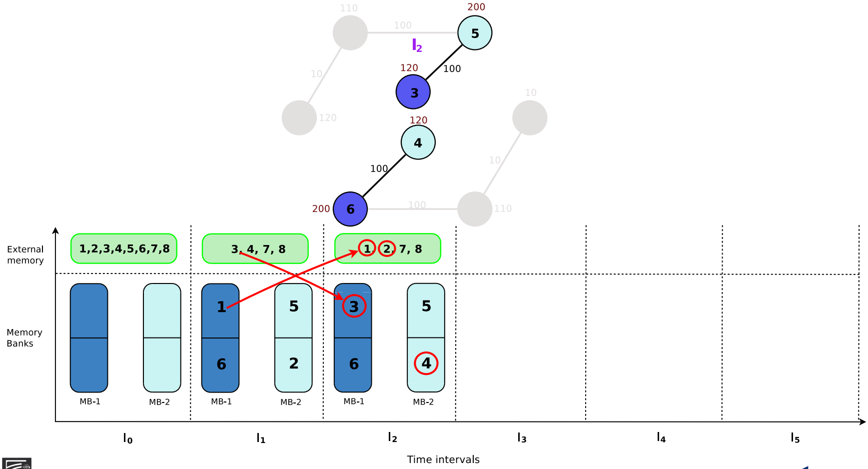
Example


External memory
1,2,3,4,5,6,7,8
Memory Banks
MB-1
MB-2
I0
I1
I2
I3
I4
I5
Time intervals

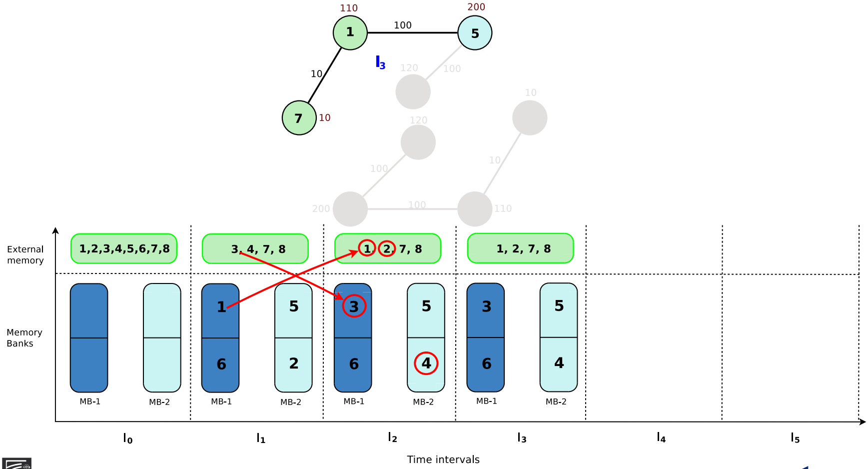
Example


110
200
1
100
5
I_1
10
120
100
10
10
120
100
10
200
6
100
2
110
I_1
External memory
1,2,3,4,5,6,7,8
3, 4, 7, 8
Memory Banks
1
5
6
2
MB-1
MB-2
MB-1
MB-2
I_0
I_1
I_2
I_3
I_4
I_5
Time intervals

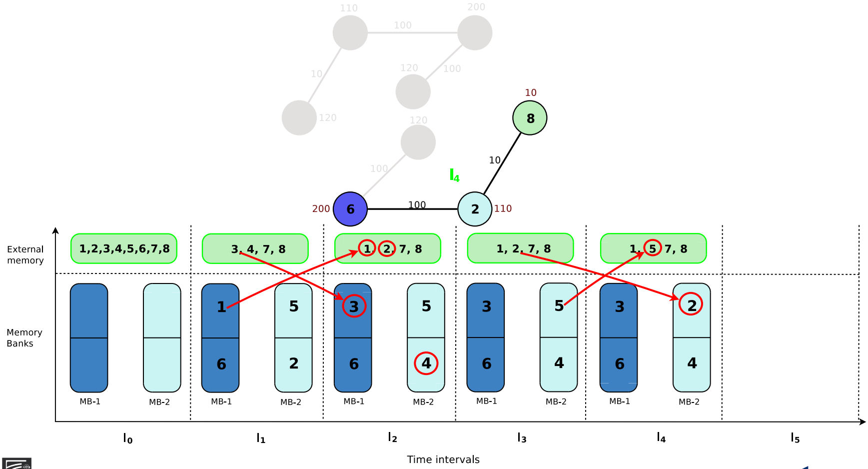
Example


110
100
200
5
I2
120
100
3
10
10
120
120
4
10
100
200
6
100
110
External memory
1,2,3,4,5,6,7,8
3, 4, 7, 8
1, 2, 7, 8
Memory Banks
1
6
5
2
3
6
5
4
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
I0
I1
I2
I3
I4
I5
Time intervals

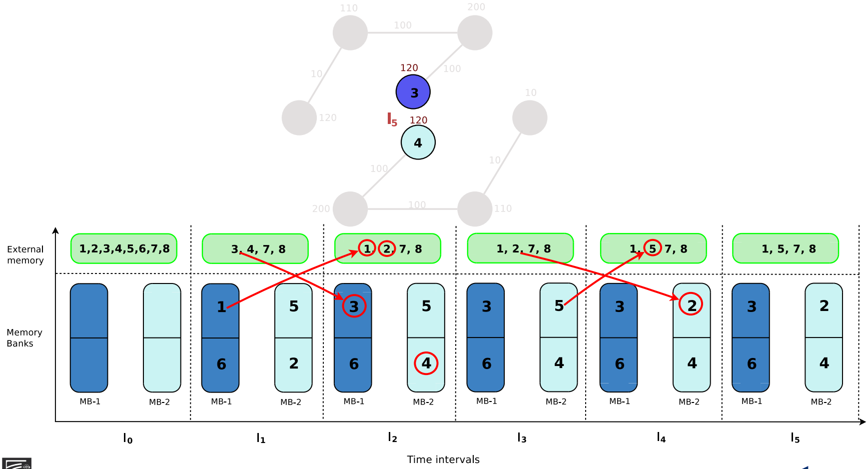
Example


110
200
1
100
5
10
7
10
I_3
120
100
10
120
10
100
200
100
110
External memory
Memory Banks
1,2,3,4,5,6,7,8
3, 4, 7, 8
1, 2, 7, 8
1, 2, 7, 8
1
6
5
2
3
6
5
4
3
6
5
4
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
I_0
I_1
I_2
I_3
I_4
I_5
Time intervals

Example


110
200
100
10
120
100
120
120
100
10
8
10
I4
200
6
100
2
110
External memory
Memory Banks
1,2,3,4,5,6,7,8
3, 4, 7, 8
1, 2, 7, 8
1, 2, 7, 8
1, 5, 7, 8
1
6
5
2
3
6
5
4
3
6
5
4
3
6
2
4
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
MB-1
MB-2
I0
I1
I2
I3
I4
I5
Time intervals

Example


110
200
100
10
100
120
3
120
I_5
120
10
4
100
10
200
100
110
External memory
Memory Banks
1,2,3,4,5,6,7,8
3, 4, 7, 8
1, 2, 7, 8
1, 2, 7, 8
1, 5, 7, 8
1, 5, 7, 8
MB-1
MB-2
I_0
I_1
I_2
I_3
I_4
I_5
Time intervals

MemExplorer Dynamic

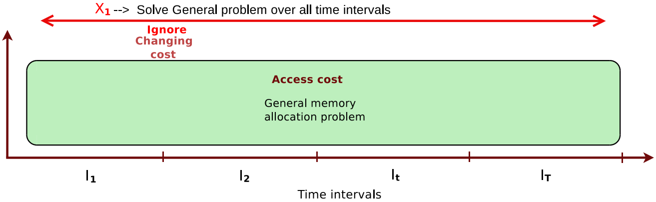
Objective

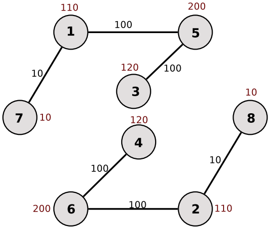
Allocate a memory bank or the external memory to any data structure of the application for **each time interval**

- minimize data access time and data moving time
- satisfy the memory banks' capacity.

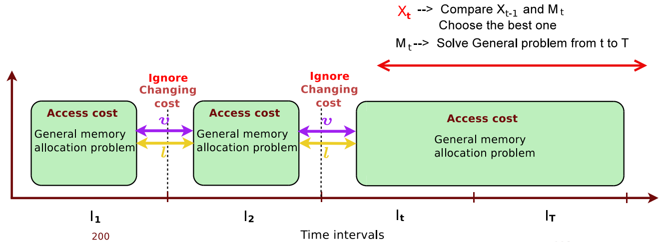
Two proposed approaches:
→ Long-term approach
→ Short-term approach

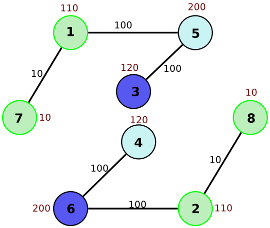
Long-term approach



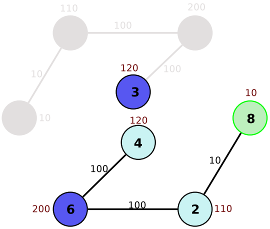


Long-term approach



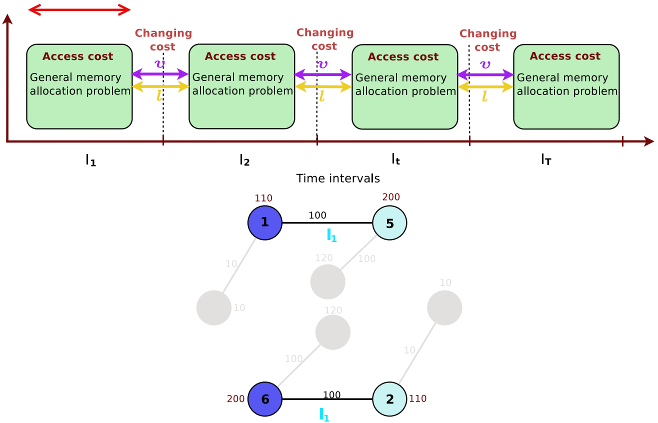


Compare
two solutions

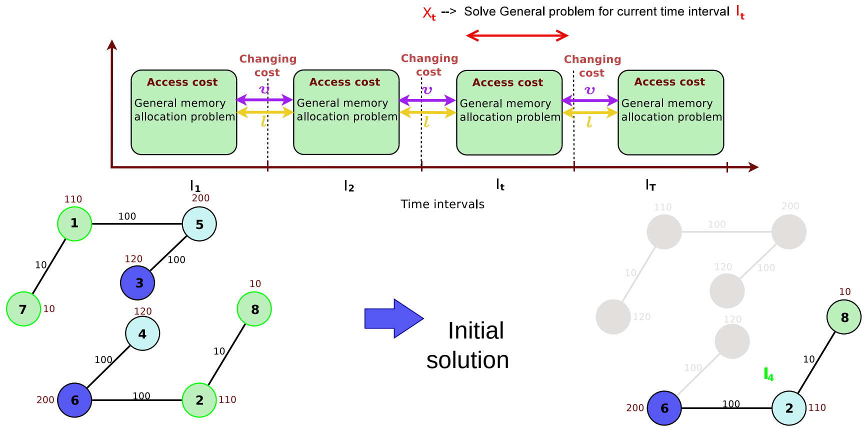


Short-term approach

X_1 --> Solve General problem for I_1



Short-term approach



Outline

Conclusions

- Four problems to memory allocation
- Approaches inspired from graph coloring algorithms
- Well balanced search in terms of intensification and diversification
- Results are better than ILP (Xpress-MP) and LocalSearch 1.0 (Bouygues e-lab. `http://e-lab.bouygues.com`)
- General memory allocation problem implemented in *SoftExplorer*

Future work

- Adapt our approaches for the memory allocations with small granularity
- Extend our approaches for other memory allocation problems

- New upper bounds exploiting graph topology
- Adapt algorithms for the Bin Packing to memory allocation problems
- Use the idea of the Knapsack algorithm for our problems
- Design matheuristics to memory allocation problems

Future work

- Mid-term approach to combine the benefits of Short term and Long term approaches
- Global approach for the dynamic memory allocation

- Continue the implementation in *SoftExplorer*

Thank you for your attention

Publications:

- Two journal articles:
  - Discrete Applied Mathematics
  - Journal of Heuristics
- Three international conferences: Evocop, EU/Meeting and CTW09
- Three national conferences: Roadef 2011, MajecSTIC and Roadef 2010