Convex clustering (work in progress)

B. Gaujal, G. Huard, J. Pecero, E. Thierry and D. Trystram
Laboratoire ID-IMAG
Denis.Trystram@imag.fr
Guillaume.Huard@imag.fr
Context and Motivation

- High-performance computing on distributed memory architectures (PC clusters and grid infrastructures).
  - High latency of interconnection network.

- The objective is to schedule a parallel application:
  - Determining where and when to execute the tasks
  - Minimize the makespan (denoted by $\omega$)

- Focus: taking into account large communication costs into the scheduling decision is a key point to reach high performance.
Classical application model: precedence task graph

- Vertices: computation tasks
- Edges: data dependencies between tasks.
The delay model

[Rayward-Smith 86, Papadimitriou and Yannakakis 90]

- Unit execution time for tasks
- Communications at a fixed cost $c$
- Simple model:
  - no overhead
  - no contention
  - total overlap
Recall of the basic scheduling problem

Instance: Precedence task graph and a delay C
Problem: Choose for each task a location and a date
Objective: minimize the makespan

NP-hard problem even for simple cases.
No constant guaranty for large communication delays!
Recall of the basic scheduling problem

Large number of heuristics, three main families:

– Extention of list algorithms:
  ETF (Earliest task first) [Hwang and al. 89]
– Locations assignment based on critical path:
  DSC (Dominant sequence clustering)
  [Gerasoulis and Yang 94]
– Graph decomposition:
  CLANS [McCreary and al. 89]
Convex clustering

- Idea:
  assign tasks to locations in convex groups
- Convexity:
  A cluster $A$ is convex iff $\forall x, z \in A, x \rightarrow y \land y \rightarrow z \Rightarrow y \in A$
Characteristics

• Advantages:
  – Schedules based on convex clusters are 2-dominant [Trystram and Lepere 2000]
  – The resulting graph is acyclic and:
    • clustering makes the grain coarser
    • classical guaranteed algorithms can be used thereafter

• Related approach:
  – CLANS [McCreary and al. 89] are special case of convex clusters
Recursive approach of the problem

- Find in graph $G=(V,E)$ two independent sets of tasks $A_1$ and $A_2$.
- Split recursively $A_1$, $A_2$, $A^>$ and $A^<$ if such a splitting allows to decrease the longest path in regard to a sequential execution.
Illustration

Graph with 16 tasks and communication delay $C = 2$. 
By construction, $\omega_R \leq |A^\prec| + 2.C + \max(|A_1|,|A_2|) + |A^\succ|$
Thus, $\omega_R \leq |V| + 2C - \min(|A_1|,|A_2|) \leq 14$
Second level of splitting
« partitioning » DAG problem

• **Instance:**
  oriented acyclic graph $G(V,E)$

• **Solution:**
  two disjoint groups of independent tasks : $A_1$ et $A_2$
such that for all task $x$ in $A_1$ and $y$ in $A_2$, there is no path between $x$ and $y$ (and vice versa)

• **Objective:**
  Maximize the size of the smallest group
  $\text{Max } (\min(|A_1|, |A_2|))$
« partitioning » DAG is NP-Complete

- Proof:
  From [Garey and Johnson 79]
  [GT24] BALANCED COMPLETE BIPARTITE SUBGRAPH
  INSTANCE: Bipartite graph $G=(V,E)$, positive integer $K \leq |V|$
  QUESTION: Are there two disjoint subsets $V_1, V_2 \subseteq V$ such that
  $|V_1| = |V_2| = K$ and such that $u \in V_1, v \in V_2$ implies that
  $\{u, v\} \in E$ ?
« partitioning » DAG is NP-Complete

Start from a bipartite graph
« partitioning » DAG is NP-Complete

Revert all its edges
« partitioning » DAG is NP-Complete

Add a direction to them
« partitioning » DAG is NP-Complete

Add proper strings of $N$ nodes
« partitioning » DAG is NP-Complete

Find a convex decomposition with both $A_1$ and $A_2$ of size $N+K$: this has to include nodes of the strings
« partitioning » DAG is NP-Complete

Find a convex decomposition with both $A_1$ and $A_2$ of size $N+K$
« partitioning » DAG is NP-Complete

This gives us our complete balanced bipartite graph
\[ \text{max } z \]
\[ \text{such that} \]
\[ z \leq \sum_{u \in V} A_1(u) \]
\[ z \leq \sum_{u \in V} A_2(u) \]
\[ \forall u \in V, A^\lt(u) + A_1(u) + A_2(u) + A^\gt(u) = 1 \]
\[ \forall e = (u, v) \in E, \ A_1(u) + A^\lt(v) < 2 \]
\[ A_2(u) + A^\lt(v) < 2 \]
\[ A^\gt(u) + A^\lt(v) < 2 \]
\[ A_1(u) + A_2(v) < 2 \]
\[ A_2(u) + A_1(v) < 2 \]
\[ A^\gt(u) + A_1(v) < 2 \]
\[ A^\gt(u) + A_2(v) < 2 \]
Algorithm for the partitioning tree problem

- By dynamic programming: compute each possible (left,right) couple of possible independant sets sizes
  - Start from leaves, label them with (1,0)
  - Compute on each node all the possible couples of size repartition
    - use a decreasing order for sizes
    - keep only the dominant couples (greater in all coordinates)
    - at most $O(n^2)$ couples to store on each node
    - proceed child by child (updating the set)
    - three cases to consider for a pair of couples (a,b) and (c,d)
      - $(a+b+c+d+1, 0)$ (fuse)
      - $(a+c, b+d)$ (combine)
      - $(a+d, b+c)$ (cross)
      ... in the proper order
Exemple of partitioning a tree
Exemple of partitioning a tree
Exemple of partitioning a tree
Exemple of partitioning a tree
Exemple of partitioning a tree
Example of partitioning a tree
Example of partitioning a tree
Algorithm for the partitioning DAG problem

Repeat (K times)
  Choose a task x
  Determine y an independent task from x
  Compute both sets $S_x = \text{succ}(x)$ and $S_y = \text{succ}(y)$
  return $A_1 = (S_x \setminus S_y)$ and $A_2 = (S_y \setminus S_x)$

Algorithm from [Trystram and Lepere 2000]
Heuristic for the partitioning DAG problem

Repeat (K times)

Choose a task $x$

Determine $y$ an independent task from $x$

Compute both sets $S_x = \text{succ}(x)$ and $S_y = \text{succ}(y)$

return $A_1 = (S_x \setminus S_y)$ and $A_2 = (S_y \setminus S_x)$
Heuristic for the partitioning DAG problem

Repeat (K times)

Choose a task $x$ 

Determine $y$ an independent task from $x$

Compute both sets $S_x = \text{succ}(x)$ and $S_y = \text{succ}(y)$

return $A_1 = (S_x \setminus S_y)$ and $A_2 = (S_y \setminus S_x)$
Heuristic for the partitioning DAG problem

Repeat (K times)

Choose a task $x$

Determine $y$ an independent task from $x$

Compute both sets $S_x = \text{succ}(x)$ and $S_y = \text{succ}(y)$

return $A_1 = (S_x \setminus S_y)$ and $A_2 = (S_y \setminus S_x)$
Heuristics for the partitioning DAG problem

Repeat (K times)

Choose a task \( x \)

Determine \( y \) an independent task from \( x \)

Compute both sets \( S_x = \text{succ}(x) \) and \( S_y = \text{succ}(y) \)

return \( A_1 = (S_x \setminus S_y) \) and \( A_2 = (S_y \setminus S_x) \)

\[
S_x \setminus S_y
\]

\[
S_y \cap S_x
\]

Cost of an itération \( O(e) \)
Conclusions

• A new approach for clustering:
  – resulting graph is acyclic
  – 2-dominant class of clustering approach

• Some possible recursive decomposition explored:
  – 4 sets division with maximal parallelism is strongly NP-Complete
  – polynomial for trees
  – previous heuristic still valid
Perspectives

- Find a better criterion for recursive decomposition
- Improve the randomized approach with genetic algorithms along with more complex criteria (J. Pecero)
- Combine non unit execution times for tasks with the transition to a coarser grain
- Find a guaranteed algorithm independant from $c$