Generalizing the Edge-Finder Rule for the Cumulative Constraint

Vincent Gingras, Claude-Guy Quimper

Abstract

We present two novel filtering algorithms for the Cumulative constraint based on a new energetic relaxation. We introduce a generalization of the Overload Check and Edge-Finder rules based on a function computing the earliest completion time for a set of tasks. Depending on the relaxation used to compute this function, one obtains different levels of filtering. We present two algorithms that enforce these rules. The algorithms utilize a novel data structure that we call Profile and that encodes the resource utilization over time. Experiments show that these algorithms are competitive with the state-of-the-art algorithms, by doing a greater filtering and having a faster runtime.

Cumulative Scheduling Problem

- **Definition**: A set of tasks need to be executed, without interruption, on a cumulative resource capacity \( C \in \mathbb{Z}^+ \)
- Properties of a non-preemptive task \( i \in I = \{1, \ldots, n\} \)
  - earliest starting time: \( \text{est}_i \in \mathbb{Z} \)
  - latest completion time: \( \text{let}_i \in \mathbb{Z} \)
  - processing time: \( p_i \in \mathbb{Z}^+ \)
  - resource consumption value: \( h_i \in \mathbb{Z}^+ \)
  - earliest completion time: \( \text{ect}_i = \text{est}_i + p_i \)
  - latest starting time: \( \text{lst}_i = \text{let}_i - p_i \)
- Generalized properties to a set of tasks \( \Omega \):
  - \( \text{est}_\Omega = \min_i \text{est}_i, \text{let}_\Omega = \max_i \text{let}_i \)
  - \( \epsilon_C = \sum_i e_i \)
- **Cumulative constraint**:
  \[ \forall i \in I \ \text{dom}(S_i) = \text{est}_i, \text{lst}_i, \sum_{i \in \Omega, S_i \subseteq S} h_i \leq C \]

Overload Check

- If the energy consumption required by a set of tasks \( \Omega \) exceeds the capacity over \( \text{est}_\Omega, \text{let}_\Omega \), then the test fails.
  \[ \exists \Omega \subseteq I : C(\text{let}_\Omega - \text{est}_\Omega) < \epsilon_C \Rightarrow \text{fail} \]
- [Fahimi et al., 2014] run the Overload Check in \( O(n) \) time

Edge-Finder

1) **Detection Phase**
   - Detects “ends before end” (\( \ll \)) temporal relation
   - [Vilím, 2009] runs the Detection Phase in \( O(n \log n) \) time
   - If a task \( i \notin \Omega \) cannot be executed along the tasks in \( \Omega \) without having any of them missing their deadline, then \( \Omega \ll i \)
   \[ e_{\Omega \cup \{i\}} > C(\text{let}_\Omega - \text{est}_\Omega) \Rightarrow \Omega \ll i \]

2) **Adjustment phase**
   - Given a precedence \( \Omega \ll i \), adjusts the lower bound of \( S_i \)
   - [Vilím, 2009] runs the Adjustment Phase in \( O(n \log n) \) time, where \( k \) is the number of distinct heights
   \[ \Omega \ll i \Rightarrow \text{est}_i \geq \max_{\Omega \subseteq \Omega'} \left\{ \text{est}_{\Omega'} + \left[ C_{\Omega'} - (C - h_i)(\text{let}_{\Omega'} - \text{est}_{\Omega'}) \right] / h_i \right\} \]

Fully-Elastic Relaxation

- Revolves around the elasticity of a task [Baptiste, Le Pape, Nuijten, 2001]
- The resource consumption of a fully elastic task can fluctuate over time
- Fully-Elastic computation of \( \text{ect}_\Omega \) [Vilím, 2009]
  \[ \text{ect}_\Omega^F = \frac{\left[ C \cdot \text{est}_\Omega + e_\Omega \right]}{\epsilon_C} \]

Generalization of known filtering rules

- Overload Check
  \[ \exists \Omega \subseteq I : \text{ect}_\Omega > \text{let}_\Omega \Rightarrow \text{fail} \]
- Edge-Finder Detection
  \[ \forall i \in I, i \in I \setminus \Omega : \text{ect}_{\Omega \cup \{i\}} > \text{let}_\Omega \Rightarrow \Omega \ll i \]
- The function \( \text{ect}_\Omega \) is NP-Hard to compute, so a relaxation is necessary
- The known Overload Check and Edge-Finder rules are based on the Fully-Elastic relaxation

Horizontally-Elastic Relaxation

- We introduce a stronger relaxation that restricts the elasticity of a task
- At any time \( t \) a task can consume between \( 0 \) and \( h_i \) units of resource
- Horizontally-Elastic computation of \( \text{ect}_\Omega \) is given by
  \[ h_{\max}(t) = \min \left( \sum_{i \in \Omega, \text{est}_i \leq t < \text{let}_i} h_i, C \right) \]
  \[ h_{\text{req}}(t) = \sum_{i \in \Omega, \text{est}_i \leq t < \text{let}_i, h_i} \]
  \[ h_{\text{cons}}(t) = \min(h_{\text{req}}(t) + ov(t - 1), h_{\max}(t)) \]
  \[ ov(t) = ov(t - 1) + \text{req}(t) - h_{\text{cons}}(t), \text{ov}(\min_{i \in \Omega} \text{est}_i) = 0 \]
  \[ \text{ect}_\Omega^H = \max\{t \mid h_{\text{cons}}(t) > 0\} + 1 \]

Examples

- **Edge-Finder Detection**
  \[ \{ (\text{est}_i, \text{let}_i, p_i, h_i) \} = \{ (0, 8, 4, 1), (0, 8, 4, 1), (3, 9, 6, 3) \}
  w = \{ (4, 8, 2, 1), (4, 8, 2, 1), (a, 4, 20, 3, 2) \} \]
- **Edge-Finder Adjustment**
  \[ \{ (\text{est}_i, \text{let}_i, p_i, h_i) \} = \{ (0, 4, 2, 1), (1, 4, 1, 3), (2, 4, 1, 1) \}
  \{ x, y, z, w \} \ll v \]

Experimental Results

Conclusion

1. We generalized the Overload Check and Edge-Finder rules (Cumulative)
2. We introduced a strong relaxation to compute \( \text{ect}_\Omega \)
3. We presented a data structure to efficiently compute \( \text{ect}_\Omega^H \)
4. We presented algorithms enforcing the Overload Check and Edge-Finder rules using our relaxation in \( O(n^2) \) time and \( O(kn^2 + n^2) \) time respectively
5. Experimental results demonstrated the effectiveness of the method

Vincent Gingras, M.Sc. Student
vincent.gingras.5@ulaval.ca