Parallelization of Discrete Optimization Problems

Advanced Topics in Parallel Processing

Kumar Chapter 11

Jan Lemeire
Dept. ETRO
November-December 2017
DOP Example: Shift-Puzzle

```
1 8 3
4 7 6
```

```
4 7 6
1 8 3
```

```
1 2 3
4 5 6
7 8
```

```
2 3 4
5 6 7
8
```
Definition DOP

- Set of feasible solutions $S$
  - Find any feasible solution

- Optional: cost function $f$ for each solution
  - Find optimal feasible solution (min or max)

- In terms of complexity of solution methods, there are two classes:
  - Problems that have efficient algorithms for finding optimal solutions.
    - ex Dijkstra
  - Problems that don’t have such efficient algorithms (NP-complete)
    - ex Traveling Salesman Problem

- Algorithms
  - Exhaustive search: computational intensive due to large set size
  - Heuristic search
Tree representation

Shift Puzzle

[Diagram of a tree representation showing initial states and moves: UP, LEFT, DOWN, RIGHT, with an ellipsis indicating continuation]
Another example: scheduling of automated warehouses
Sometimes a feasible solution is OK, in other cases the optimal solution should be found.
Sequential Tree Search Algorithms

- **Depth – First search**
  1. Simple Backtracking
  2. Branch and Bound: limit the depth

- **Breadth – First search**
  1. Iterative Deepening: with open node list

- **Heuristic search**
  - **Best first search:**
    - based on breadth-first
    - With heuristic function that identifies promising nodes
Parallelization possibilities

**Algorithms**
- Brute force
- Branch & Bound
- Iterated Local Search
- ...

**Levels of Parallelism**
- Alternative algorithms
- Alternative scenarios (due to uncertainty)
- Multi-search
- Tree-level parallelism
- Node-level parallelism (neighbor exploration)
- Parallel calculation of score or bound

**Parallel Hardware**
- Cluster
- Multicore
- Accelerator

**Suitability is problem-dependent!**

Our focus here

Discrete Optimization Problems
Jan Lemeire
Parallel Depth-first Tree Search

Distribution of tree:

After expansion of root node: send children & problem to slaves

CPU₁  CPU₂  CPU₃
DOP looking for a feasible solution
Parallel Work Anomalies

Sequential work ≠ Parallel work!

\[ T_{seq} \neq \sum_{i=1}^{p} T_{work}^i \]

\[ T_{seq} + T_{anomaly} = \sum_{i}^{p} T_{work}^i \]

Search Overhead Factor = \( \frac{\text{parallel work}}{\text{sequential work}} = \frac{\sum_{i=1}^{p} T_{work}^i}{T_{seq}} \)

In our approach: considered as overhead (can be positive or negative)

Impact on overhead: \( \frac{T_{anomaly}}{T_{seq}} \)
Parallel Overhead

- Partitioning: low
- Communication: low
- Synchronization: no
- Returning results: low
- **Idling: HIGH**
  - Due to load imbalances
  - **Solution**: dynamic load balancing
    - “when finished, ask for work”

Find solutions for:
1. Donor Selection
2. Termination Detection
Load imbalances

![Diagram](a) and (b)

**Figure 11.7** The unstructured nature of tree search and the imbalance resulting from static partitioning.
Donor selection

- **Asynchronous Round Robin**
  Each processor keeps a cyclic list

- **Global Round Robin**
  Master keeps a cyclic list

- **Random polling**
  Random selection of donor
Termination detection

- Via master (centralized)
  - For example: if donor selection happens via master

- Dijkstra’s token algorithm (distributed/local)
  - Arrange processes in a ring
  - Without DLB: a simple token that is passed around by processes when they are terminated
  - With DLB:
    - A boolean per process: keeps track whether work has been redistributed since last pass of token
    - A boolean as token: keeps track whether work has been redistributed by one process and the token should go around again

- Tree-Based (partly-distributed)

- other ...
**Tree-based Termination Detection**

*Idea:* Associates weights with individual work pieces

- Master starts with weight 1.
- It sends work to \((p-1)\) slaves together with weight \(1/p\) for each. It keeps weight \(1/p\).
- If a slave finishes: its sends its weight to master. Master adds it to its weight.
- If a slave sends a part of its work, it sends halve of its weight.
- Termination when weight at master becomes one and master has finished.
Parallel Best-first Search

**Breadth-first**: similar to depth-first (every process explores part of the tree)

**Best-first**: Keep stack of *open nodes*, ordered by a heuristic function

1. **Centralised strategy**: keep stack on master
   ⇒ send part of nodes to slaves
   ⇒ Slaves return expanded nodes
   A LOT OF COMMUNICATION

2. **Distributed/local strategy**: stack on each processor
   Synchronization of open node list necessary:
   - Random communication strategy
   - Ring communication strategy
   - Blackboard communication strategy
Ring communication

Figure 11.15  A message-passing implementation of parallel best-first search using the ring communication strategy.
Blackboard Strategy

**Centralized**

**Figure 11.16** An implementation of parallel best-first search using the blackboard communication strategy.

Discrete Optimization Problems

*Jan Lemeire*
Graph Representation

If the same states can be encountered through different paths (cf puzzle)


*Disadvantage of a tree representation*: nodes will be checked multiple times!


**Solution**: Keep a closed node list
Check every expanded node whether already visited
 - If parallel: synchronization of list (as for open node list)
 - retrieve node with a hash function