**Experiment No - 10**

**Aim**: Study of Learning Basic Block Scheduling Heuristics from Optimal Data.

**Date:**

**Competency and Practical Skills:**

* Knowledge of basic computer architecture concepts
* Understanding of compiler design and optimization techniques
* Ability to analyze and interpret performance data
* Proficiency in programming languages such as C/C++ and assembly language

**Relevant CO:** CO4

**Objectives:**

By the end of this experiment, the students should be able to:

* + Understanding the concept of basic block scheduling and its importance in compiler optimization.
  + Understanding the various heuristics used for basic block scheduling.
  + Analyzing optimal data to learn the basic block scheduling heuristics.
  + Comparing the performance of the implemented basic block scheduler with other commonly used basic block schedulers.

**Software/Equipment:** Computer system

**Theory:**

Instruction scheduling is an important step for improving the performance of object code produced by a compiler. Basic block scheduling is important in its own right and also as a building block for scheduling larger groups of instructions such as superblocks. The basic block instruction scheduling problem is to find a minimum length schedule for a basic block a straight-line sequence of code with a single-entry point and a single exit point subject to precedence, latency, and resource constraints. Solving the problem exactly is known to be difficult, and most compilers use a greedy list scheduling algorithm coupled with a heuristic. The heuristic is usually hand-crafted, a potentially time-consuming process. Modern architectures are pipelined and can issue multiple instructions per time cycle. On such processors, the order that the instructions are scheduled can significantly impact performance.

The basic block instruction scheduling problem is to find a minimum length schedule for a basic block a straight-line sequence of code with a single entry point and a single exit point subject to precedence, latency, and resource constraints. Instruction scheduling for basic blocks is known to be NP-complete for realistic architectures. The most popular method for scheduling basic blocks continues to be list scheduling.

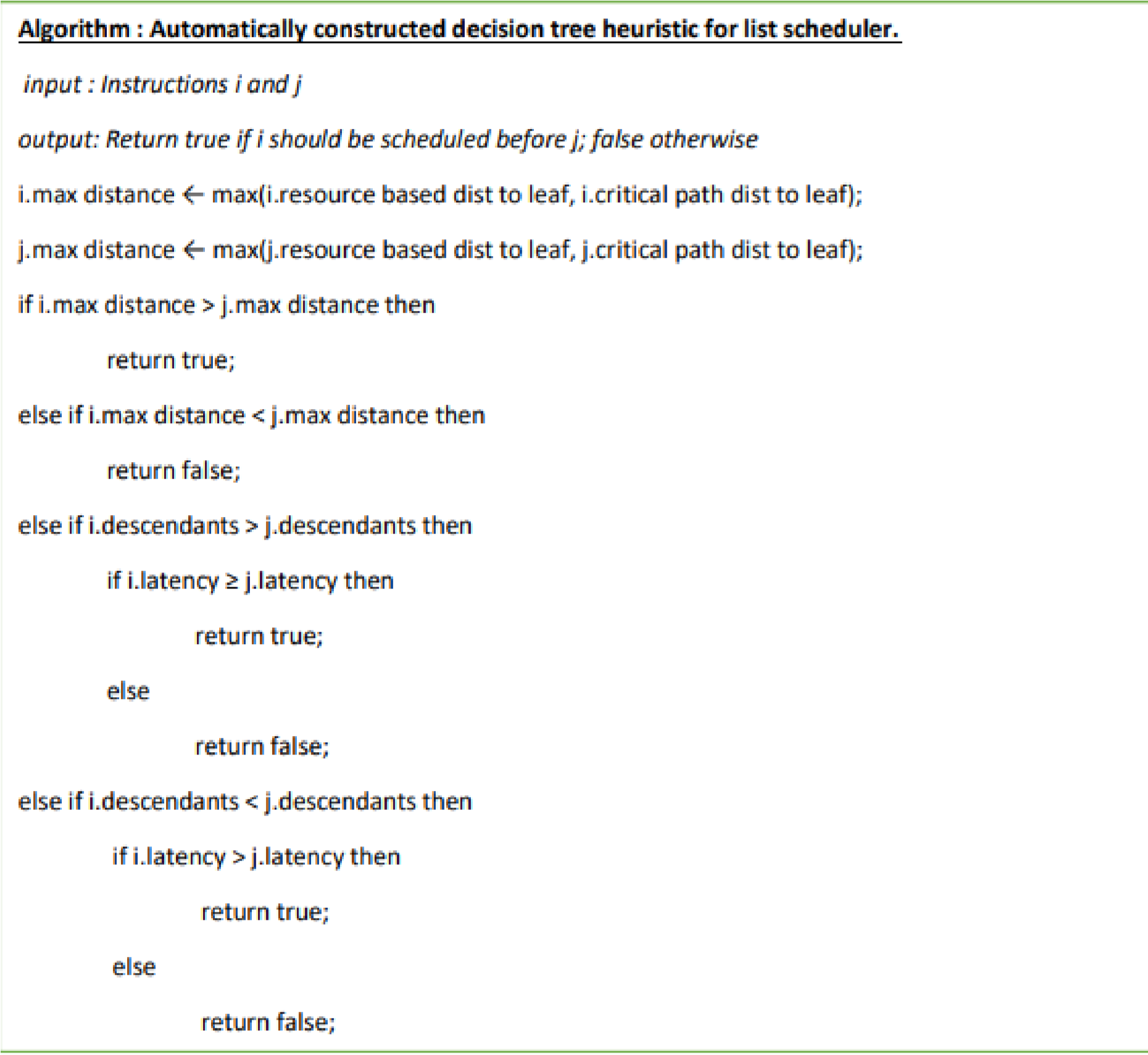
For e.g.: We consider multiple-issue pipelined processors. On such processors, there are multiple functional units and multiple instructions can be issued (begin execution) each clock cycle. Associated with each instruction is a delay or latency between when the instruction is issued and when the result is available for other instructions which use the result. In this paper, we assume that all functional units are fully pipelined and that instructions are typed. Examples of types of instructions are load/store, integer, floating point, and branch instructions. We use the standard labelled directed acyclic graph (DAG) representation of a basic block (see Figure 1(a)). Each node corresponds to an instruction and there is an edge from i to j labelled with a positive integer l (i, j) if j must not be issued until i has executed for l (i, j) cycles. Given a labelled dependency DAG for a basic block, a schedule for a multiple-issue processor specifies an issue or start time for each instruction or node such that the latency constraints are satisfied and the resource constraints are satisfied. The latter are satisfied if, at every time cycle, the number of instructions of each type issued at that cycle does not exceed the number of functional units that can execute instructions of that type. The length of a schedule is the number of cycles needed for the schedule to complete; i.e., each instruction has been issued at its start time and, for each instruction with no successors, enough cycles have elapsed that the result for the instruction is available. The basic block instruction scheduling problem is to construct a schedule with minimum length.

Instruction scheduling for basic blocks is known to be NP-complete for realistic architectures. The most popular method for scheduling basic blocks continues to be list scheduling. A list scheduler takes a set of instructions as represented by a dependency DAG and builds a schedule using a bestfirst greedy heuristic. A list scheduler generates the schedule by determining all instructions that can be scheduled at that time step, called the ready list, and uses the heuristic to determine the best instruction on the list. The selected instruction is then added to the partial schedule and the scheduler determines if any new instructions can be added to the ready list.

The heuristic in a list scheduler generally consists of a set of features and an order for testing the features. Some standard features are as follows. The path length from a node i to a node j in a DAG is the maximum number of edges along any path from i to j. The critical-path distance from a node i to a node j in a DAG is the maximum sum of the latencies along any path from i to j. Note that both the path length and the critical-path distance from a node i to itself is zero. A node j is a descendant of a node i if there is a directed path from i to j; if the path consists of a single edge, j is also called an immediate successor of i. The earliest start time of a node i is a lower bound on the earliest cycle in which the instruction i can be scheduled.

In supervised learning of a classifier from examples, one is given a training set of instances, where each instance is a vector of feature values and the correct classification for that instance, and is to induce a classifier from the instances. The classifier is then used to predict the class of instances that it has not seen before. Many algorithms have been proposed for supervised learning. One of the most widely used is decision tree learning. In a decision tree the internal nodes of the tree are labelled with features, the edges to the children of a node are labelled with the possible values of the feature, and the leaves of the tree are labelled with a classification. To classify a new example, one starts at the root and repeatedly tests the feature at a node and follows the appropriate branch until a leaf is reached. The label of the leaf is the predicted classification of the new example.

**Algorithm:**



This document is on automatically learning a good heuristic for basic block scheduling using supervised machine learning techniques. The novelty of our approach is in the quality of the training data we obtained training instances from very large basic blocks and we performed an extensive and systematic analysis to identify the best features and to synthesize new features— and in our emphasis on learning a simple yet accurate heuristic.

**Observations and Conclusion:**

* Instruction scheduling is an important step for improving the performance of object code produced by a compiler.
* Basic block scheduling is important as a building block for scheduling larger groups of instructions such as superblocks.
* The basic block instruction scheduling problem is to find a minimum length schedule for a basic block a straight-line sequence of code with a single-entry point and a single exit point subject to precedence, latency, and resource constraints.
* Solving the problem exactly is known to be difficult, and most compilers use a greedy list scheduling algorithm coupled with a heuristic.
* Modern architectures are pipelined and can issue multiple instructions per time cycle. The order that the instructions are scheduled can significantly impact performance.
* Instruction scheduling for basic blocks is known to be NP-complete for realistic architectures.
* The most popular method for scheduling basic blocks continues to be list scheduling, which takes a set of instructions as represented by a dependency DAG and builds a schedule using a best-first greedy heuristic.
* The heuristic in a list scheduler generally consists of a set of features and an order for testing the features.
* Decision tree learning is one of the most widely used algorithms for supervised learning.

**Quiz:**

1. What is the basic block instruction scheduling problem?
2. Why is instruction scheduling important for improving the performance of object code produced by a compiler?
3. What are the constraints that need to be considered in solving the basic block instruction scheduling problem?
4. What is the most popular method for scheduling basic blocks?
5. What is the heuristic used in a list scheduler?

**Suggested Reference:**

1. <https://dl.acm.org/doi/10.5555/1105634.1105652>
2. https://www.worldcat.org/title/1032888564

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Knowledge**  **(2)** | | **Problem**  **Recognition**  **(2)** | | **Documentati on (2)** | | **Presentation**  **(2)** | | **Ethics (2)** | | **Total** |
| **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** |
| **Marks** |  |  |  |  |  |  |  |  |  |  |  |