Method and system for splitting scheduling problems into sub-problems

US 20140136252 A1

Publication number	US20140136252 A1
Publication type	Application
Application number	US 13/655,934
Publication date	May 15, 2014
Filing date	Oct 19, 2012
Priority date	Oct 31, 2011
Also published as	CN104662566A, WO2013066902A2, WO2013066902A3
Inventors	David Everton Norman
Original Assignee	Applied Materials, Inc.
Export Citation	BiBTeX, EndNote, RefMan

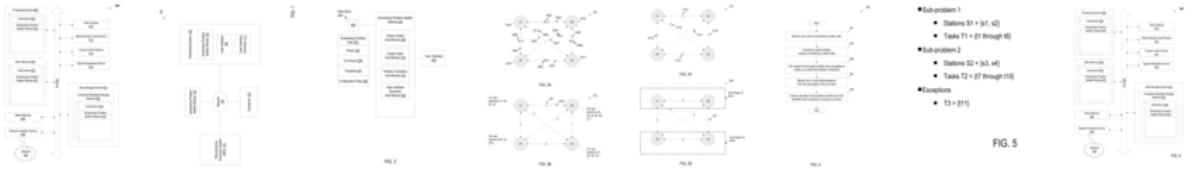
Patent Citations (9), Non-Patent Citations (11), Classifications (3), Legal Events (1)

External Links: USPTO, USPTO Assignment, Espacenet

ABSTRACT

A computing system receives user input of scheduling problem data. The scheduling problem data relates to a scheduling problem and includes one or more stations and tasks to be performed by at least one station. The computing system constructs a graph problem using the scheduling problem data. The graph problem includes a graph. The computing system cuts the graph into sub-graphs using a cut algorithm to create a cut result that satisfies a threshold and identifies one or more task exceptions from the sub-graphs in the cut result. The one or more task exceptions are tasks that can be assigned to more than one sub-graph. The computing system creates scheduling sub-problems pertaining to the one or more task exceptions using the cut result.

IMAGES (8)



DESCRIPTION

TECHNICAL FIELD

[0001] Implementations of the present disclosure relate to scheduling generally, and more particularly, to splitting scheduling problems into sub-problems.

BACKGROUND

[0002] A difficulty in addressing a manufacturing scheduling problem can be related to the problem size. Typical manufacturing scheduling problems involve a large number of stations and a significant number of tasks to be performed on the stations. For example, scheduling can depend on a number of tools, a number of lots, a sequential order of operations, constraints, etc. Traditional scheduling systems spend a great amount of time and computing resources in solving a scheduling problem that involves many variables and factors. The difficulty grows very fast as the size of the scheduling problem grows. For this reason, large scheduling problems can be impossible to solve directly.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The present disclosure is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that different references to “an” or “one” implementation in this disclosure are not necessarily to the same implementation, and such references mean at least one.

[0004] FIG. 1 is a block diagram illustrating a scheduling system utilizing a scheduling problem splitter module.

CLAIMS (20)

1. A method comprising:

receiving user input of scheduling problem data, the scheduling problem data relating to a scheduling problem and comprising one or more stations and a plurality of tasks to be performed by at least one station;

constructing a graph problem using the scheduling problem data, the graph problem comprising a graph;

cutting the graph into sub-graphs using a cut algorithm to create a cut result that satisfies a threshold;

identifying one or more task exceptions from the sub-graphs in the cut result, wherein the one or more task exceptions is a task that can be assigned to more than one sub-graph; and

creating, by a computing system, a plurality of scheduling sub-problems pertaining to the at least one task exception using the cut result.

2. The method of claim 1, wherein the cut that satisfies the threshold comprises the cut resulting in a fewest number of task exceptions, the cut resulting in a fewest number of types of tasks, or the cut resulting in a fewest number of task recipes being exceptions.

3. The method of claim 1, wherein the graph comprises a plurality of nodes representing stations pertaining to the scheduling problem and one or more lines connecting pairs of the plurality of nodes to

splitter module.

FIGS. 3A-D illustrate example graphs representing a scheduling problem.

FIG. 4 illustrates one implementation of a method for splitting a scheduling problem into scheduling sub-problems.

FIG. 5 illustrates example scheduling sub-problems and an example task exception.

FIG. 6 illustrates an example computer system.

DETAILED DESCRIPTION

Implementations of the disclosure are directed to a method and system for splitting a scheduling problem into scheduling sub-problems. A computing system receives user input of scheduling problem data. The scheduling problem data identifies stations and tasks to be performed by at least one station. The computing system constructs a graph problem using the scheduling problem data. The graph problem can include a graph, which the computing system can partition into sub-graphs using a cut algorithm to create a cut result that satisfies a threshold. Examples of a threshold can include, and are not limited to, creating a cut result that has the fewest task exceptions, creating a cut result that has the fewest number of types of tasks, creating a cut result that has the fewest number of task recipes that are exceptions. The computing system identifies task exceptions from the sub-graphs. A task exception can be a task that can be assigned to more than one sub-graph. The computing system creates scheduling sub-problems using the cut result. Implementations greatly reduce the amount of time and resources used to solve a manufacturing scheduling problem by automatically splitting a large scheduling problem into sub-problems and identifying one or more task exceptions associated with the sub-problems.

FIG. 1 is a block diagram illustrating a manufacturing system **100** including a fabrication system data source (e.g., manufacturing execution system (MES) **101**), a dispatcher **103**, and a scheduling system **105** communicating, for example, via a network. **120**. The network **120** can be a local area network (LAN), a wireless network, a mobile communications network, a wide area network (WAN), such as the Internet, or similar communication system.

In one implementation, the scheduling system **105** includes a scheduling problem splitter module **107**. In another implementation, the scheduling system **105** communicates with an external scheduling problem splitter module **107**, for example, via the network **120**. The MES **101**, dispatcher **103**, scheduling system **105**, and scheduling problem splitter module **107** can be individually hosted by any type of computing device including server computers, gateway computers, desktop computers, laptop computers, tablet computer, notebook computer, PDA (personal digital assistant), mobile communications devices, cell phones, smart phones, hand-held computers, or similar computing device. Alternatively, any combination of MES **101**, dispatcher **103**, scheduling system **105**, and scheduling problem splitter module **107** can be hosted on a single computing device including server computers, gateway computers, desktop computers, laptop computers, mobile communications devices, cell phones, smart phones, hand-held computers, or similar computing device.

A scheduling system **105** can receive input relating to a scheduling problem from a scheduling problem splitter module **107** to create a schedule of when one or more tasks can be performed and on which stations (tool). A schedule can be a list of tasks that each station processes. The schedule can include the task start times. Stations can be certified to run certain tasks. A task can be a task used in the manufacturing of semiconductors and there can be different types of tasks. Examples of tasks can include, and are not limited to, a task to manufacture a product, a task to use a reticle manufacturing tool, a

nodes.

4. The method of claim 1, wherein constructing the graph further comprises:

assigning a weight to at least one of a node in the graph representing a station or a line in the graph connecting a pair of nodes, wherein the line represents at least one task relating to the stations corresponding to the connected pair of nodes, and the weight indicates at least one of a number of tasks associated with the stations corresponding to the pair of nodes, a percentage of tasks associated with the stations corresponding to the pair of nodes, or a task priority.

5. The method of claim 1, further comprising:

sending data indicating the plurality of scheduling sub-problems to a scheduling system to schedule tasks times based on the data.

6. The method of claim 1, wherein the plurality of tasks are processed in a specified flow order.

7. The method of claim 1, wherein the plurality of tasks relate to manufacturing semiconductors.

8. A non-transitory computer readable storage medium including instructions that, when executed by a processing device, cause the processing device to perform a method comprising:

receiving user input of scheduling problem data, the scheduling problem data relating to a scheduling problem and comprising one or more stations and a plurality of tasks to be performed by at least one station;

constructing a graph problem using the scheduling problem data, the graph problem comprising a graph;

cutting the graph into sub-graphs using a cut algorithm to create a cut result that satisfies a threshold;

identifying one or more task exceptions from the sub-graphs in the cut result, wherein the one or more task exceptions is a task that can be assigned to more than one sub-graph; and

creating, by the processing device, a plurality of scheduling sub-problems pertaining to the at least one task exception using the cut result.

9. The non-transitory computer readable storage medium of claim 8, wherein the cut that satisfies the threshold comprises the cut resulting in a fewest number of task exceptions, the cut resulting in a fewest number of types of tasks, or the cut resulting in a fewest number of task recipes being exceptions.

10. The non-transitory computer readable storage medium of claim 8, wherein the graph comprises a plurality of nodes representing stations pertaining to the scheduling problem and one or more lines connecting pairs of the plurality of nodes to represent tasks relating to the stations corresponding to the connected nodes.

11. The non-transitory computer readable storage medium of claim 8, wherein constructing the graph further comprises:

assigning a weight to at least one of a node in the graph representing a station or a line in the graph connecting a pair of nodes, wherein the line represents at least one task relating to the stations corresponding to the connected pair of nodes, and the weight indicates at least one of a number of tasks associated with the stations corresponding to the pair of nodes, a percentage of

wafers, etc.

The scheduling problem splitter module **107** can identify a scheduling problem and automatically split the scheduling problem into smaller scheduling sub-problems. A scheduling problem can involve a set of tasks T and a set (e.g., set S) of stations, also known as tools. For example, a scheduling problem may involve forty to fifty stations and more than two thousand tasks. Each task can be processed on one or more stations. The scheduling problem splitter module **107** can split a scheduling problem if there exist disjoint subsets of tasks, such as task subset T**1** and task subset T**2**, such that T is the union of task subset T**1** and task subset T**2**, and similar disjoint subsets of stations, such as station subset S**1** and station subset S**2**, such that set S is the union of station subset S**1** and station subset S**2**, and where the tasks in task subset T**1** can only process on stations in station subset S**1** and tasks in task subset T**2** can only process on stations in station subset S**2**. The scheduling problem splitter module **107** can automatically convert a manufacturing scheduling problem into a graph theory problem and can apply a cut algorithm to the graph theory problem to solve the problem. The scheduling problem splitter module **107** can convert the solution from the graph theory problem format back into a format for the scheduling problem, where the scheduling problem can be represented by smaller sub-problems and one or more task exceptions associated with the sub-problems. The scheduling problem splitter module **107** can provide data reflecting the smaller scheduling sub-problems to the scheduling system **105**, which can use the data to provisionally schedule the times for various tasks to be performed. In one implementation, the scheduling system **105** is coupled to a factory system data source (e.g., MES **101**, ERP) to receive lot data and equipment status data and uses the scheduling sub-problem data, lot data, and equipment status data to provisionally schedule tasks to be performed. In one implementation, scheduling system **105** can include a graphical user interface (GUI) generator **111** to create and provide a user interface **109** (e.g., GUI) to a user (e.g., an industrial engineer). User interface **109** can enable a user (e.g., an industrial engineer) to model a provisional schedule. In one implementation, the scheduling system **105** provides the entire schedule to a dispatcher **103**. The dispatcher **103** can be integrated through the MES **101** to dispatch, for example, wafer lots accordingly.

FIG. 2 is a block diagram of one implementation of a scheduling problem splitter module **200**. In one implementation, the scheduling problem splitter module **200** can be the same as the scheduling problem splitter module **107** of FIG. 1. The scheduling problem splitter module **200** can include a graph creator sub-module **205**, a graph cutter sub-module **210**, a problem translator sub-module **215**, and a user interface (UI) generator sub-module **225**.

The user interface generator sub-module **225** can generate a user interface **202** that can receive a set of scheduling problem data **251** as user input for a scheduling problem. The scheduling problem splitter module **200** can also receive a set of scheduling problem data **251** or a portion of the scheduling problem data **251** from another system in a manufacturing system. The scheduling problem data **251** can define tasks to be performed (e.g., a set of tasks T) and a set (e.g., set S) of stations or tools for performing the tasks. The scheduling problem data **251** can also describe whether a task can be processed on one or more stations and can identify one or more stations for performing the task. The scheduling problem data **251** can be stored in a data store **250** that is coupled to the scheduling problem splitter module **200**. There can be multiple sets of scheduling problem data **251** for multiple scheduling problems stored in the data store **250**. The data store **250** can be a persistent storage unit. A persistent storage unit can be a local storage unit or a remote storage unit. Persistent storage units

nodes, or a task priority.

12. The non-transitory computer readable storage medium of claim 8, further comprising:

sending data indicating the plurality of scheduling sub-problems to a scheduling system to schedule tasks times based on the data.

13. The non-transitory computer readable storage medium of claim 8, wherein the plurality of tasks relate to manufacturing semiconductors.

14. A system comprising:

a memory; and

a processing device coupled with the memory to:

receive user input of scheduling problem data, the scheduling problem data relating to a scheduling problem and comprising one or more stations and a plurality of tasks to be performed by at least one station;

construct a graph problem using the scheduling problem data, the graph problem comprising a graph;

cut the graph into sub-graphs using a cut algorithm to create a cut result that satisfies a threshold;

identify one or more task exceptions from the sub-graphs in the cut result, wherein the one or more task exceptions is a task that can be assigned to more than one sub-graph; and

create a plurality of scheduling sub-problems pertaining to the at least one task exception using the cut result.

15. The system of claim 14, wherein the cut that satisfies the threshold comprises the cut resulting in a fewest number of task exceptions, the cut resulting in a fewest number of types of tasks, or the cut resulting in a fewest number of task recipes being exceptions.

16. The system of claim 14, wherein the graph comprises a plurality of nodes representing stations pertaining to the scheduling problem and one or more lines connecting pairs of the plurality of nodes to represent tasks relating to the stations corresponding to the connected nodes.

17. The system of claim 14, wherein constructing the graph further comprises:

assigning a weight to at least one of a node in the graph representing a station or a line in the graph connecting a pair of nodes, wherein the line represents at least one task relating to the stations corresponding to the connected pair of nodes, and the weight indicates at least one of a number of tasks associated with the stations corresponding to the pair of nodes, a percentage of tasks associated with the stations corresponding to the pair of nodes, or a task priority.

18. The system of claim 14, further comprising:

sending data indicating the plurality of scheduling sub-problems to a scheduling system to schedule tasks times based on the data.

19. The system of claim 14, wherein the plurality of tasks are processed in a specified flow order.

20. The system of claim 14, wherein the plurality of tasks relate to manufacturing semiconductors.

unit, electronic storage unit (main memory) or similar storage unit. Persistent storage units can be a monolithic device or a distributed set of devices. A 'set', as used herein, refers to any positive whole number of items.

The graph creator sub-module **205** can use the scheduling problem data **251** for a particular scheduling problem to convert the scheduling problem format to a graph problem format. The graph creator sub-module **205** can construct a graph problem to represent the scheduling problem. The graph creator sub-module **205** can create a graph **253** for the graph problem. A graph **253** can be stored in a data store **250**. There can be graphs **253** for various scheduling problems stored in the data store **250**. A graph **253** can be a mathematical structure used to model pair wise relations between objects from a certain collection. A graph **253** can refer to a collection of vertices or nodes and a collection of edges that connect pairs of vertices (nodes). An edge is also hereinafter referred to as a line. A graph **253** can include graph properties, for example, and not limited to, each station (e.g., s**1**, s**2**, . . . sn) in the set S of stations for the scheduling problem can be a node in the graph, and there can be an edge (line) in the graph **253** connecting the nodes corresponding to a pair of stations to indicate that there is at least one task that can run on the pair of stations.

The scheduling problem data **251** can include one or more weight types and/or weight values, for example, as defined by user input. The graph creator sub-module **205** can assign a weight to each pair of connected stations in a graph **253** indicating, for example, the importance that the stations are connected based on, such as, constraints pertaining to the stations. The graph creator sub-module **205** can assign a weight to a line in a graph **253** based on the scheduling problem data **251**. Examples of weight types that can be calculated can include, and are not limited to, the percentage of tasks that can run on both of the connected stations, where the percentage is weighted based on the importance of the task (e.g., task priority, for example, based on user input), the percentage of operations or route steps that can be processed on both of the connected stations, the percentage of operations or route steps that can be processed on both of the connected stations, weighted by the percentage of expected lots for that operation, and a percentage indicating the importance of one task in relation to another (e.g., task **1** should be followed by task **2**). Implementations apply both to job shop scheduling, where each task is independent, and to flow shop scheduling, where tasks represent multiple operations for a single manufacturing lot and where the tasks for a lot should be processed in a certain order. The weight value assigned to a line can reflect a flow shop schedule to indicate the importance of adjacent operations (also known as a precedent constraint). The graph creator sub-module **205** can further adjust a weight based on other parameters, such as, and not limited to, physical distance between the stations, physical similarity between the stations, brand or station type.

FIG. 3A is an example graph **300** for a scheduling problem. The scheduling problem data for the scheduling problem may include stations s**1**, s**2**, s**3**, and s**4**, and tasks t**1**-t**11**. The graph **300** may include a node (e.g., nodes **301**A-D) for each station. The graph **300** may include lines (e.g., lines **303**A-F) that connect pairs of nodes indicating there is some task that can run on both stations corresponding to the connected nodes. The lines **303**A-F can be assigned based on the scheduling problem data. For example, the scheduling problem data may specify that the task t**1** can be processed on station s**1**; the tasks t**2**, t**3**, and t**4** can be processed on stations s**1** and s**2**; the tasks t**5** and t**6** can be processed on stations **2**; the task t**7** can be processed on station s**3**; the tasks t**8** and t**9** can be processed on stations s**3** and s**4**; the task t**10** can be processed on station s**4**; the task t**11** can be processed on stations s**2** and s**3**. Based on this data, for example, there may be a line **303**A in the graph **300** connecting node **301**A, which may represent station s**1**, to node **30**B, which may represent station s**2**, to indicate that there is at least one task (e.g., t**2**, t**3**, t**4**) that may run on both stations s**1** and s**2**. The lines **303**A-F can also be assigned weights (e.g., **305**A-F). In one implementation, a line weight indicates the number of tasks that can be processed on both of the stations corresponding to the nodes connected by the line. FIG. 3B is an example graph **350** having weights assigned based on the number of tasks that can be processed on both of the stations. For example, a value of '3' may be assigned as the weight **307** to line **303**A to indicate that both stations, s**1** and s**2**, can process three tasks (e.g., t**2**, t**3**, and t**4**) as defined by the scheduling problem data. In another implementation, a line weight indicates a flow shop schedule to reflect the importance of adjacent operations.

Returning to FIG. 2, the graph cutter sub-module **210** can partition a graph **253** into sub-graphs to split a scheduling problem into scheduling sub-problems to create a cut result **255** that satisfies a threshold. Examples of a threshold can include, and are not limited to, creating a cut result **255** that has the fewest task exceptions, creating a cut result **255** that has the fewest number of types of tasks, creating a cut result **255** that has the fewest number of task recipes that are exceptions. A task exception can be a task that can be performed on stations in more than one sub-graph. For example, a task exception may be a task that can be performed by stations in station subset S**1** and stations in station subset S**2**. The graph cutter sub-module **210** can cut lines of a graph **253** by applying a cut algorithm to create a cut result **255**. The cut result **255** can represent sub-graphs of a graph **253**, which has been partitioned. The cut result **255** can be stored in the data store **250**. A cut can be a partition of the vertices (nodes) of a graph **253** into two disjoint subsets. The size of a cut C=(S,T) can be the number of lines in the cut-set. If the lines are weighted, the value of the cut can be the sum of the weights. In one implementation, the graph cutter sub-module **210** applies a minimal cut algorithm to the graph **253**, which is an algorithm that makes a minimum cut to cut the graph **253** into two sub-graphs (e.g., G**1** and G**2**). Various cut algorithms to split a graph into any number of sub-graphs can be used. A cut can be a minimum cut if the cut set

(e.g., two elements per line that is cut), and a cut across two lines may result in four elements. The cut across the two lines may be the minimum cut. The graph cutter sub-module **210** can cut lines based on total weight. FIG. 3C is an example graph **370** for a scheduling problem that has lines cut based on total weight of the lines, according to various implementations. The graph **370** may have line weights (e.g., weights **371**A-F) based on the number of tasks that can be processed on both of the stations connected by a line. The cut result from cutting the lines based on the total weight may be, for example, two sub-graphs G**1** and G**2**. The graph cutter sub-module can cut the lines that have the smallest total weight. For example, lines **303**B,D,E,F may be cut. FIG. 3D is an example graph **390** for a scheduling problem that may be split into sub-graph G**1** **391**A and sub-graph G**2** **391**B after cutting the lines that have the smallest total weight (e.g., line connecting s**1** and s**4**, line connecting s**2** and s**3**, line connecting s**1** and s**3**, line connecting s**4** and s**2**).

For each sub-graph (sub-graph G**1** **391**A and sub-graph G**2** **391**B), the graph cutter sub-module **210** can identify a subset (e.g., S**1**, S**2**) of the set of stations (e.g., S), where each station subset has a corresponding subset of tasks (task subset). For example, in FIG. 3D, the graph cutter sub-module **210** may identify that the subset S**1** of stations corresponds to sub-graph G**1** **391**A. The station subset S**1** may include stations s**1** and s**2**. The graph cutter sub-module **210** can also identify that the subset of stations S**2** may correspond to sub-graph G**2** **391**B. The subset station S**2** may include stations s**3** and s**4**. The graph cutter sub-module **210** can identify that station subset S**1** may have a corresponding task subset T**1**, and that station subset S**2** may have a corresponding task subset T**2**. The task subset T**1** may include tasks t**1**-t**6**. The task subset T**2** may include tasks t**7**-t**10**.

The graph cutter sub-module **210** can also use the sub-graphs in a cut result **255** to identify one or more tasks exceptions. The one or more task exceptions can be grouped to form another task subset (e.g., T**3**). For example, the graph cutter sub-module **210** may determine that if the graph **390** is split into sub-graph G**1** **391**A and sub-graph G**2** **391**B, then there may be only one task exception. The task exception can be represented by the weight value '1' **373** which may represent task **11** (t**11**) that can be processed on s**2** (station **2**) and s**3** (station **3**). For example, since task t**11** can be processed on stations that are in different sub-graphs, task t**11** is a task exception.

The problem translator sub-module **215** can identify sub-problems for a scheduling problem using the sub-graphs, station subset for each sub-graph, and task subset for each sub-graph. The problem translator sub-module **215** can convert the sub-graphs into the format of sub-problems, for example, using a template **257** and/or configuration data **259** that is stored in the data store **250**. The problem translator sub-module **215** can group a station subset and the task subset that corresponds to the station subset as a sub-problem. For example, station subset S**1** and task subset T**1** may be grouped to form Sub-Problem **1**, and station subset S**2** and task subset T**2** may be grouped to form Sub-Problem **2**. The problem translator sub-module **215** can also identify task exceptions associated with the sub-problems for the scheduling problem. For example, the problem translator sub-module **215** may identify task t**11** as a task exception associated with Sub-Problem **1** and Sub-Problem **2**. The problem translator sub-module **215** can create another set of tasks (e.g., T**3**) to represent the one or more task exceptions. The smallest number of task exceptions can be an indication of optimal sub-graphs. For example, sub-graph G**1** **391**A and sub-graph G**2** **391**B may result in one task exception. In this example, the sub-graphs G**1** and G**2** may indicate the optimal cut of the graph.

FIG. 4 is a flow diagram of an implementation of a method **400** for splitting a scheduling problem into scheduling sub-problems. Method **400** can be performed by processing logic that can comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (e.g., instructions run on a processing device), or a combination thereof. In one implementation, method **400** is performed by the scheduling problem splitter module **107** hosted by a computing device of FIG. 1.

At block **401**, processing logic receives user input of scheduling problem data. The scheduling problem data can include data received from another system in a manufacturing system. The user input for the scheduling problem data can include, for example, and is not limited to, the number stations, station data (e.g., status) describing specific stations, the tasks to be performed (e.g., a set of tasks T), a set (e.g., set S) of stations or tools for performing the tasks, weight type, and weight values. The weight types and weight values can be for stations and/or tasks. The scheduling problem data can also describe whether a task can be processed on one or more stations and can identify which stations can perform the task.

At block **403**, processing logic constructs a graph problem using the scheduling problem data. Processing logic can apply the weights, if any, from the user input to construct a weighted graph problem. At block **405**, processing logic cuts the graph into sub-graphs by applying a cut algorithm to create a cut result that satisfies a threshold. Examples of a threshold can include and are not limited to, creating a cut result that has the fewest task exceptions, creating a cut result that has the fewest number of types of tasks, creating a cut result that has the fewest number of task recipes that are exceptions. Processing logic selects an optimal split of sub-graphs that satisfies the threshold. The threshold can be a user-defined threshold. Processing logic can apply different edge (line) weights when constructing the graph. The different weights can cause different types of exceptions to be minimized. At block **407**, processing logic identifies one or more task exceptions from the sub-graphs in the cut result. At block **409**, processing logic creates scheduling sub-problems pertaining to the one or more task exceptions using the cut result. Subsequently, processing logic may provide the sub-problems and task

exception to a sub-problem. For example, the heuristic may be based on the bandwidth of the sub-problems. In another example, a user (e.g., process engineer) can be notified, for example, via a scheduling system, of the one or more task exceptions and can assign a task exception to a sub-problem. FIG. 5 illustrates example scheduling sub-problems and an example task exception for the scheduling problem described by the graphs in FIGS. 3A-D. For example, a process engineer may determine that Sub-problem **1** does not have enough work and may assign the task exception task **11** to Sub-problem **1**.

Returning to FIG. 4, portions of method **400** can be an iterative method. The number of iterations can be based on a configurable and/or user-defined value. For example, processing logic can perform multiple iterations until, for instance, a minimal number of task exceptions are identified. For example, processing logic may initially split a graph (e.g., graph **350** in FIG. 3B), where Sub-graph **1** may include stations s**1** and s**4**, and Sub-graph **2** may include stations s**2** and s**3**, at block **405**. With such a split, processing logic may identify, for example, five task exceptions. For example, tasks t**2**, t**3**, and t**4** may be run on stations s**1** and s**2**, and tasks t**8** and t**9** may be run on stations s**4** and s**1**. Subsequently, processing logic may split the graph, where Sub-graph **1** may include stations s**1** and s**2**, and Sub-graph **2** may include stations s**4** and s**3** at block **405**, and identifies, for example, one task exception (e.g., task t**11** can be run on stations s**2** and s**3**). Processing logic can compare the number of task exceptions from the various splits and can select the split resulting in the fewest number of task exceptions at block **405**. For example, processing logic may select the split, where Sub-graph **1** may include stations s**1** and s**2**, and Sub-graph **2** may include stations s**4** and s**3**, since this particular split results in one task exception.

FIG. 6 is a block diagram illustrating an example computing device **600**. In one implementation, the computing device corresponds to a computing device hosting a scheduling problem splitter module **107** of FIG. 1. The computing device **600** includes a set of instructions for causing the machine to perform any one or more of the methodologies discussed herein. In alternative implementations, the machine may be connected (e.g., networked) to other machines in a LAN, an intranet, an extranet, or the Internet. The machine may operate in the capacity of a server machine in client-server network environment. The machine may be a personal computer (PC), a set-top box (STB), a server, a network router, switch or bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term “machine” shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein.

The exemplary computer device **600** includes a processing system (processing device) **602**, a main memory **604** (e.g., read-only memory (ROM), flash memory, dynamic random access memory (DRAM) such as synchronous DRAM (SDRAM), etc.), a static memory **606** (e.g., flash memory, static random access memory (SRAM), etc.), and a data storage device **618**, which communicate with each other via a bus **608**.

Processing device **602** represents one or more general-purpose processing devices such as a microprocessor, central processing unit, or the like. More particularly, the processing device **602** may be a complex instruction set computing (CISC) microprocessor, reduced instruction set computing (RISC) microprocessor, very long instruction word (VLIW) microprocessor, or a processor implementing other instruction sets or processors implementing a combination of instruction sets. The processing device **602** may also be one or more special-purpose processing devices such as an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), a digital signal processor (DSP), network processor, or the like. The processing device **602** is configured to execute the scheduling problem splitter module **670** for performing the operations and steps discussed herein.

The computing device **600** may further include a network interface device **608**. The computing device **600** also may include a video display unit **610** (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT)), an alphanumeric input device **612** (e.g., a keyboard), a cursor control device **614** (e.g., a mouse), and a signal generation device **616** (e.g., a speaker).

The data storage device **618** may include a computer-readable storage medium **628** on which is stored one or more sets of instructions (instructions of scheduling problem splitter module **670**) embodying any one or more of the methodologies or functions described herein. The scheduling problem splitter module **670** may also reside, completely or at least partially, within the main memory **604** and/or within the processing device **602** during execution thereof by the computing device **600**, the main memory **604** and the processing device **602** also constituting computer-readable media. The scheduling problem splitter module **670** may further be transmitted or received over a network **620** via the network interface device **608**.

While the computer-readable storage medium **628** is shown in an example implementation to be a single medium, the term “computer-readable storage medium” should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term “computer-readable storage medium” shall also be taken to include any medium that is capable of storing, encoding or carrying a set of instructions for execution by the machine and that cause the machine to perform any one or more of the methodologies of the present disclosure. The term “computer-readable storage medium” shall accordingly be taken to include, but not be limited to, solid-state memories, optical media, and magnetic media.

art having the benefit of this disclosure, that implementations of the disclosure may be practiced without these specific details. In some instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the description.

Some portions of the detailed description are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as “receiving,” “constructing,” “cutting,” “identifying,” “creating,” “assigning,” “sending,” or the like, refer to the actions and processes of a computing device, or similar electronic computing device, that manipulates and transforms data represented as physical (e.g., electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage devices.

Implementations of the disclosure also relate to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but not limited to, any type of disk including optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions.

It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other implementations will be apparent to those of skill in the art upon reading and understanding the above description. The scope of the disclosure should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

PATENT CITATIONS

Cited Patent	Filing date	Publication date	Applicant	Title
US6112023 *	Feb 17, 1998	Aug 29, 2000	Lucent Technologies Inc.	Scheduling-based hardware-software co-synthesis of heterogeneous distributed embedded systems
US8812653 *	Aug 5, 2010	Aug 19, 2014	Novell, Inc.	Autonomous intelligent workload management
US20030167265 *	Jun 7, 2001	Sep 4, 2003	Corynen Guy Charles	Computer method and user interface for decision analysis and for global system optimization
US20060291390 *	Jun 22, 2005	Dec 28, 2006	Bin Zhang	Determination of a state of flow through or a cut of a parameterized network
US20080218518 *	Mar 6, 2007	Sep 11, 2008	Zhou Yunhong	Balancing collections of vertices in a network
US20090047677 *	Jul 28, 2008	Feb 19, 2009	The Arizona Board of Regents, a body corporate of the State of Arizona acting for & on behalf of	Methods for generating a distribution of optimal solutions to nondeterministic polynomial optimization problems
US20090313596 *	Jun 11, 2008	Dec 17, 2009	Bernhard Lippmann	System and Method for Integrated Circuit Planar Netlist Interpretation
US20110029982 *	Jul 30, 2009	Feb 3, 2011	Bin Zhang	Network balancing procedure that includes redistributing flows on arcs incident on a batch of vertices
US20110313984 *	Jul 23, 2010	Dec 22, 2011	Palo Alto Research Center Incorporated	System and method for parallel graph searching utilizing parallel edge partitioning

* Cited by examiner

NON-PATENT CITATIONS

Reference		
1	*	Ascheuer, N. et al., A Cutting Plane Approach to the Sequential Ordering Problem (With Applications to Job Scheduling in Manufacturing), SIAM J. Optimization, Vol. 3, No. 1, February 1993
2	*	Ji, Xiaoyun, Graph Partition Problems with Minimum Size ConstraintsRenssler Polytechnic Institute, November 2004

Reference		
4	*	Lu, Yufeng, Scheduling of Wafer Test Processes in Semiconductor ManufacturingVirginia Polytechnic Institute and State University, October 26, 2001
5	*	Mohring, Rolf H. et al., Solving Project Scheduling Problems by Minimum Cut ComparisonsManagement Science, Vol. 29, No. 3, March 2003
6	*	Nemhauser, G.L., A Cutting Plane Algorithm for the Single Machine Scheduling Problem with Release TimesCombinatorial Optimization, NATO ASI Series, Vol. 82, 1992
7	*	Pacciarelli, Dario, Parallel machine scheduling in a flexible manufacturing systemINFOR, Vol. 29, No. 2, May 2001
8	*	Rios-Mercado, Roger Z. et al., Computational Experience with a Branch-and-Cut Algorithm for Flowshop Scheduling with Setups, University of Texas at Austin, May 1997
9	*	Schlogel, Kirk Andrew, Graph Partitioning for Emerging Scientific SimulationsUniversity of Minesota, November 1999
10	*	Song, Yang et al., Bottleneck Station Scheduling in Semiconductor Assembly and Test Manufacturing Using Ant Colony Optimization, IEEE Transactions on Automation Science and Engineering, Vol. 4, No. 4, October 2007
11	*	Stecco, Gabriella et al., A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times, Computers & Operations Research, Vol. 35, 2008

* Cited by examiner

CLASSIFICATIONS

U.S. Classification	705/7.13
International Classification	G06Q10/06
Cooperative Classification	G06Q10/06311

LEGAL EVENTS

Date	Code	Event	Description
Oct 19, 2012	AS	Assignment	**Owner name:** APPLIED MATERIALS, INC., CALIFORNIA **Free format text:** ASSIGNMENT OF ASSIGNORS INTEREST;ASSIGNOR:NORMAN, DAVID EVERTON;REEL/FRAME:029160/0211 **Effective date:** 20121019