SiPS/SiPSi Version 1.08p4 Reference Manual

Shunji Tanaka

Dec 6, 2022

Contents

1 Introduction **7**
- 1.1 Single-Machine Scheduling Problem . . . 7
- 1.2 Restriction . . . 8

2 Quick Start Guide **9**
- 2.1 Header File . . . 11
- 2.2 Create an Instance . . . 11
- 2.3 Specify Instance Data . . . 12
  - 2.3.1 Specify by functions . . . 12
  - 2.3.2 Read from a data file . . . 12
  - 2.3.3 Set the user-defined job cost function . . . 12
- 2.4 Solve the Instance . . . 13
- 2.5 Access the Solution . . . 13
  - 2.5.1 Retrieve the solution . . . 13
  - 2.5.2 Display the solution . . . 13
- 2.6 Free the Problem . . . 13

3 API Reference **15**
- 3.1 Create/Modify the Instance . . . 15
  - 3.1.1 SiPS_create_problem . . . 15
  - 3.1.2 SiPS_create_problem_with_name . . . 15
  - 3.1.3 SiPS_read_problem . . . 15
  - 3.1.4 SiPS_create_problem_from_file . . . 16
  - 3.1.5 SiPS_free_problem . . . 16
  - 3.1.6 SiPS_TWT_set_job . . . 16
  - 3.1.7 SiPS_TWT_set_jobs . . . 17
  - 3.1.8 SiPS_TWET_set_job . . . 18
  - 3.1.9 SiPS_TWET_set_jobs . . . 18
  - 3.1.10 SiPS_delete_job . . . 18
  - 3.1.11 SiPS_delete_jobs . . . 19
  - 3.1.12 SiPS_delete_all_jobs . . . 19
  - 3.1.13 SiPS_set_cost_function . . . 19
  - 3.1.14 SiPS_set_cost_function_long . . . 20
  - 3.1.15 SiPS_set_horizon . . . 20
  - 3.1.16 SiPS_set_cost_type . . . 21
  - 3.1.17 SiPS_set_initial_sequence . . . 21
- 3.2 Access the Instance . . . 22
  - 3.2.1 SiPS_get_problem_size . . . 22

3.2.2 SiPS_get_job . . . 22
3.2.3 SiPS_get_jobs . . . 23
3.2.4 SiPS_print_jobs . . . 23
3.3 Solver Parameters . . . 23
3.3.1 SiPS_set_int_param . . . 23
3.3.2 SiPS_set_real_param . . . 24
3.4 Solve the Instance . . . 24
3.4.1 SiPS_solve . . . 24
3.5 Access the Solution . . . 24
3.5.1 SiPS_get_solution . . . 24
3.5.2 SiPS_get_cputime . . . 25
3.5.3 SiPS_print_solution . . . 25

4 Available Parameters **27**
4.1 Common Parameters . . . 27
4.1.1 SIPS_I_VERBOSE (integer) . . . 27
4.1.2 SIPS_R_EPS (double) . . . 27
4.1.3 SIPS_R_FUNCEPS (double) . . . 27
4.1.4 SIPS_R_TIMELIMIT (double) . . . 27
4.1.5 SIPS_I_MAXMEMORY (integer) . . . 27
4.1.6 SIPS_I_WARMSTART (integer) . . . 27
4.1.7 SIPS_I_DUMMYJOBS (integer) . . . 28
4.1.8 SIPS_I_SEARCHTYPE (integer) . . . 28
4.1.9 SIPS_I_DPSIZE (integer) . . . 28
4.1.10 SIPS_I_TIEBREAK (integer) . . . 28
4.2 Stages 1 and 2 . . . 28
4.2.1 SIPS_I_MINUPDATE (integer) . . . 28
4.3 Stage 1 . . . 29
4.3.1 SIPS_I_SKIPSTAGE1 (integer) . . . 29
4.3.2 SIPS_R_MULTIPLIER (double) . . . 29
4.3.3 SIPS_R_INITSTEP1 (double) . . . 29
4.3.4 SIPS_R_MAXSTEP1 (double) . . . 29
4.3.5 SIPS_I_SHRINKITER1 (integer) . . . 29
4.3.6 SIPS_R_SHRINK1 (double) . . . 29
4.3.7 SIPS_R_EXPAND1 (double) . . . 29
4.3.8 SIPS_R_INITTERMITER1 (double) . . . 29
4.3.9 SIPS_R_TERMITER1 (double) . . . 29
4.3.10 SIPS_R_TERMRATIO1 (double) . . . 30
4.3.11 SIPS_I_UBUPDATE1 (integer) . . . 30
4.4 Stage 2 . . . 30
4.4.1 SIPS_R_INITSTEP2 (double) . . . 30
4.4.2 SIPS_R_MAXSTEP2 (double) . . . 30
4.4.3 SIPS_I_SHRINKITER2 (integer) . . . 30
4.4.4 SIPS_R_SHRINK2 (double) . . . 30
4.4.5 SIPS_R_EXPAND2 (double) . . . 30
4.4.6 SIPS_R_INITTERMITER2 (double) . . . 30
4.4.7 SIPS_R_TERMITER2 (double) . . . 30
4.4.8 SIPS_R_TERMRATIO2 (double) . . . 31
4.4.9 SIPS_I_UBITERATION (integer) . . . 31
4.5 Stage 3 . . . 31

4.5.1 SIPS_I_SECSWITCH (integer) . . . 31
4.5.2 SIPS_R_SECINIRATIO (double) . . . 31
4.5.3 SIPS_R_SECRATIO (double) . . . 31
4.5.4 SIPS_I_BISECTION (integer) . . . 31
4.5.5 SIPS_I_MAXMODIFIER (integer) . . . 31
4.5.6 SIPS_I_MODSWITCH (integer) . . . 31
4.5.7 SIPS_I_ZEROMULTIPLIER (integer) . . . 32
4.5.8 SIPS_I_UBUPDATE3 (integer) . . . 32

5 Miscellany **33**
5.1 Data File Format . . . 33
5.1.1 SiPS . . . 33
5.1.2 SiPSi . . . 33
5.2 Front-Ends sips/sipsi . . . 34
5.2.1 Usage . . . 34
5.2.2 Options . . . 34

Chapter 1

Introduction

SiPS (Single-machine scheduling Problem Solver) and SiPSi (Single-machine scheduling Problem Solver with idle time) are solvers for general single-machine scheduling to minimize total job completion cost. They are currently offered as C libraries. For the detailed algorithms, please refer to our paper:

> S. Tanaka, S. Fujikuma, and M. Araki (2009). An exact algorithm for single-machine scheduling without machine idle time, Journal of Scheduling, vol. 12, no. 6, pp. 575–593. DOI:10.1007/s10951-008-0093-5.
>
> S. Tanaka and S. Fujikuma (2012). A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time, Journal of Scheduling. vol. 15, no. 3, pp. 347–361. DOI:10.1007/s10951-011-0242-0.

SiPS/SiPSi is distributed under the **2-clause BSD license**. It is not a license condition, but I am pleased if you will refer to the above papers when you write a paper using SiPS/SiPSi because the URL offering the program may change in the future. Also, donation is welcome. ;-)

My contact address is

Shunji Tanaka
Department of Electrical Engineering, Kyoto University
Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan
tanaka@kuee.kyoto-u.ac.jp

1.1 Single-Machine Scheduling Problem

Consider that n jobs (job 1, job 2, ..., job n) are to be processed on a single machine. Each job $i \in \mathcal{N} = \{1, 2, \ldots, n\}$ is given an integer processing time p_i, an integer release date r_i and a cost function $f_i(t)$. The machine can process at most one job at a time and no preemption is allowed. Denote the completion time of job i by $C_i (\geq r_i + p_i)$ and assume that C_i is integral. The objective is to find a schedule that minimizes the total job completion cost $\sum_{i \in \mathcal{N}} f_i(C_i)$. According to the standard classification, this problem is referred to as $1||\sum f_i(C_i)$ when $r_i = 0$ for all $i \in \mathcal{N}$, or otherwise $1|r_i|\sum f_i(C_i)$.

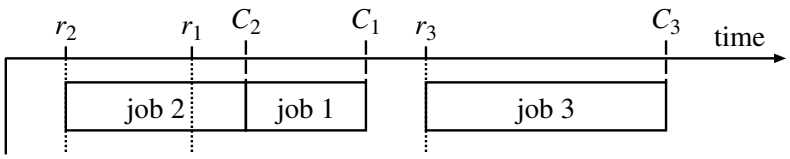

Figure 1.1: Gantt Chart of a Schedule

A schedule is often denoted by a Gantt chart, whose horizontal axis represents the the elapsed time. A simple example is shown in Figure 1.1.

SiPSi is applicable to $1||\sum f_i(C_i)$ and $1|r_i|\sum f_i(C_i)$. On the other hand, SiPS is only to the problem without idle time, i.e. $1||\sum f_i(C_i)$ without idle time, which is denoted by $1|\text{noidle}|\sum f_i(C_i)$.

Ordinary, the cost function $f_i(t)$ is specified by the 3-tuple (d_i, α_i, β_i): an integer due date d_i, an earliness weight α_i and a tardiness weight β_i. More specifically, the cost function $f_i(t)$ is given by

$$\begin{aligned} f_i(t) &= \alpha_i \max\{d_i - C_i, 0\} + \beta_i \max\{C_i - d_i, 0\} \\ &= \max\{\alpha_i(d_i - C_i), \beta_i(C_i - d_i)\}. \end{aligned} \tag{1.1}$$

Therefore, if (d_i, α_i, β_i)=(0, 0, w_i), the objective function becomes the total weighted completion time $\sum_{i\in\mathcal{N}} w_i C_i$ and if (d_i, α_i, β_i)=(d_i, 0, w_i), it becomes the total weighted tardiness $\sum_{i\in\mathcal{N}} w_i T_i$.

1.2 Restriction

There is a major restriction in this version. SiPS and SiPSi cannot be used (linked) simultaneously because names of the interface functions are common. (This can be easily resolved, but...) Please choose one of the two based on what you want to solve. SiPSi is more general but slower than SiPS when it is applied to $1|\text{noidle}|\sum f_i(C_i)$. Moreover, SiPSi is not applicable to $1|\text{noidle}|\sum f_i(C_i)$ when $f_i(C_i)$ is nonregular: $f_i(t)$ is not a nondecreasing function of t.

By default, α_i and β_i must also be integral. However, this restriction can be removed at the cost of speed by defining "`COST_REAL`" and recompiling the program. If the objective value is integral but can take a very large value, define "`COST_LONGLONG`" and recompile the program. In this case, it is better to define "`REAL_LONG`" at the same time.

SiPSi does not work properly when real-valued cost functions are specified by SiPS_set_cost_function(). It is because the upper bound computation algorithm assumes the cost functions to be piecewise linear.

The completion time C_i must also be integral, but this restriction does not make any problems in most cases.

Chapter 2

Quick Start Guide

```
 1:  /* header file */
 2:  #ifndef SIPSI /* for SiPS */
 3:  #include "sips.h"
 4:  #else /* for SiPSi */
 5:  #include "sipsi.h"
 6:  #endif
 7:
 8: cost_t f(int, int);
 9:
10: int main(void)
11: {
12:   int ret;
13:   /* create the problem */
14:   sips *prob = SiPS_create_problem();
15:
16:   /* specify job data */
17: #ifndef SIPSI /* for SiPS */
18:   /*                      No.  name   p   d  ew  tw */
19:   SiPS_TWET_set_job(prob,   0, "J1", 10, 10,  3,  1);
20:   SiPS_TWET_set_job(prob,   1, "J2", 20, 55,  2,  2);
21:   SiPS_TWET_set_job(prob,   2, "J3", 30, 40,  1,  3);
22: #else /* for SiPSi */
23:   /*                      No.  name   p   r   d  ew  tw */
24:   SiPS_TWET_set_job(prob,   0, "J1", 10,  0, 10,  3,  1);
25:   SiPS_TWET_set_job(prob,   1, "J2", 20,  5, 55,  2,  2);
26:   SiPS_TWET_set_job(prob,   2, "J3", 30, 10, 40,  1,  3);
27: #endif
28:
29:   /* read from file */
30:   /* SiPS_read_problem(prob, "sample.dat"); */
```

```
31:
32:   /* specify cost function */
33:   /* SiPS_set_cost_function(prob, f); */
34:   /* specify scheduling horizon (SiPSi) */
35:   /* SiPS_set_horizon(prob, 55 + 10 + 20 + 30); */
36:
37:   /* solve the problem */
38:   ret = SiPS_solve(prob);
39:
40:   /* retrieve the solution */
41:   /* if(ret == SIPS_SOLVED) { */
42:   /*   int job[3], C[3]; */
43:   /*   cost_t f; */
44:   /*   SiPS_get_solution(prob, &f, job, C); */
45:   /* } */
46:
47:   /* just display the solution */
48:   if(ret == SIPS_SOLVED) {
49:     SiPS_print_solution(prob);
50:   }
51:
52:   /* release the problem  */
53:   SiPS_free_problem(prob);
54:
55:   return(ret);
56:  }
57:
```

```
58:  #define max(a, b) (((a)>(b))?(a):(b))
59:
60:  cost_t f(int i, int t)
61:  {
62:    cost_t v;
63:
64:    switch (i) {
65:    default:
66:    case 0:
67:      v = 3*max(10-t, 0) + 1*max(t-10, 0);
68:      break;
69:    case 1:
70:      v = 2*max(55-t, 0) + 2*max(t-55, 0);
71:      break;
72:    case 2:
73:      v = 1*max(40-t, 0) + 3*max(t-40, 0);
74:      break;
75:    }
76:
77:    return(v);
78:  }
```

Figure 2.1: Sample Program

This chapter gives a quick start guide for SiPS/SiPSi. A typical program using SiPS/SiPSi is shown in Figure 2.1.

2.1 Header File

For SiPS:

```
3: #include "sips.h"
```

For SiPSi:

```
5: #include "sipsi.h"
```

Please include either "`sips.h`" (SiPS) or "`sipsi.h`" (SiPSi).

CAUTION: These header files cannot be included simultaneously.

2.2 Create an Instance

```
14: sips *prob = SiPS_create_problem();
```

SiPS_create_problem() creates and returns a pointer to a problem object (a structure named "sips"). This object keeps all the information on the instance and the solution.

2.3 Specify Instance Data

There are two ways of specifying an instance: (1) specify instance data by functions, and (2) read from a data file.

2.3.1 Specify by functions

For SiPS:

```
19:  /*                        No.  name   p   d  ew  tw */
20:  SiPS_TWET_set_job(prob,    0, "J1", 10, 10,  3,  1);
21:  SiPS_TWET_set_job(prob,    1, "J2", 20, 55,  2,  2);
22:  SiPS_TWET_set_job(prob,    2, "J3", 30, 40,  1,  3);
```

For SiPSi:

```
23:  /*                        No.  name   p   r   d  ew  tw */
24:  SiPS_TWET_set_job(prob,    0, "J1", 10,  0, 10,  3,  1);
25:  SiPS_TWET_set_job(prob,    1, "J2", 20,  5, 55,  2,  2);
26:  SiPS_TWET_set_job(prob,    2, "J3", 30, 10, 40,  1,  3);
```

SiPS_TWET_set_job() sets job data one by one. This function takes seven arguments for SiPSi: job number, job name, processing time, release date, due date, earliness weight and tardiness weight. In the case of SiPS, the release date cannot be specified and the number of the arguments is six.

2.3.2 Read from a data file

```
30:  SiPS_read_problem(prob, "sample.dat");
```

Another way to specify an instance is to read from a file. SiPS_read_problem() reads instance data from the specified file. The file format is described in Section 5.1.

2.3.3 Set the user-defined job cost function

```
33:  SiPS_set_cost_function(prob, f);
34:  /* specify scheduling horizon (SiPSi) */
35:  SiPS_set_horizon(prob, 55 + 10 + 20 + 30);
```

```
60:  cost_t f(int i, int t)
```

If you want to use a job cost function that cannot be expressed only by its due date, earliness weight and tardiness weight, use SiPS_set_cost_function(). In this case, due dates and weights specified by SiPS_TWET_set_job() are ignored. (To be more precise, due dates are exploited in the tie-breaking rule.) The cost function takes two arguments, job number i and time t, and returns $f_i(t)$. In the case of SiPSi, the end of the scheduling horizon must also be specified by SiPS_set_horizon().

2.4 Solve the Instance

```
38:   ret = SiPS_solve(prob);
```

SiPS_solve() solves the current instance. If the instance is optimally solved, it returns `SIPS_SOLVED`.

2.5 Access the Solution

2.5.1 Retrieve the solution

```
44:   SiPS_get_solution(prob, &f, job, C);
```

SiPS_get_solution() returns the information of the obtained solution. The objective value is substituted into `f`, and the processing order and the completion time are into the arrays `job` and `C`, respectively. That is, `job[0]` is the job number of the first job, `C[0]` is the completion time of the first job, `job[1]` is the job number of the second job, `C[1]` is the completion time of the second job, and so on.

2.5.2 Display the solution

```
49:     SiPS_print_solution(prob);
```

If you only want to display the solution, use SiPS_print_solution().

2.6 Free the Problem

```
53:   SiPS_free_problem(prob);
```

SiPS_free_problem() releases the allocated memory space.

Chapter 3

API Reference

3.1 Create/Modify the Instance

3.1.1 SiPS_create_problem

Synopsis

```
sips *SiPS_create_problem(void);
```

Description

This function creates a problem object and returns a pointer to it.

3.1.2 SiPS_create_problem_with_name

Synopsis

```
sips *SiPS_create_problem_with_name(char *name);
```

Description

This function is the same as SiPS_create_problem() except that the argument is used as the problem name.

3.1.3 SiPS_read_problem

Synopsis

```
int SiPS_read_problem(sips *prob, char *filename);
```

Description

This function reads instance data from a file `filename` and stores them in `prob`.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.4 SiPS_create_problem_from_file

Synopsis

```
sips *SiPS_create_problem_from_file(char *filename);
```

Description

This function creates a problem object from a file `filename` and returns a pointer to it. In other words, it executes SiPS_create_problem() and SiPS_read_problem() at the same time.

Returns

When it fails to read instance data from a file `filename`, it returns `NULL`.

3.1.5 SiPS_free_problem

Synopsis

```
int SiPS_free_problem(sips *prob);
```

Description

This function releases all the allocated memory for the problem object `prob`.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.6 SiPS_TWT_set_job

Synopsis

For SiPS:

```
int SiPS_TWT_set_job(sips *prob, int no, char *name,
                     int p, int d, cost_t w);
```

For SiPSi:

```
int SiPS_TWT_set_job(sips *prob, int no, char *name,
                     int p, int r, int d, cost_t w);
```

Description

This function specifies the data of a single job.

`prob` : problem object
`no` : number (starts from 0)
`name` : name (can be `NULL`)
`p` : processing time
`r` : release date (SiPSi only)
`d` : due date
`w` : tardiness weight

It assumes that the objective is total weighted tardiness, and the earliness weight is automatically set to 0. If, for example, job 2 is specified after job 0 without job 1, the data of job 1 is automatically generated. If an existing job is specified, its data are overriden by the specified data.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.7 SiPS_TWT_set_jobs

Synopsis

For SiPS:

```
int SiPS_TWT_set_jobs(sips *prob, int n, int *no, char **name,
                      int *p, int *d, cost_t *w);
```

For SiPSi:

```
int SiPS_TWT_set_jobs(sips *prob, int n, int *no, char **name,
                      int *p, int *r, int *d, cost_t *w);
```

Description

This function specifies the data of several jobs at the same time.

`prob` : problem object
`n` : number of jobs to be specified
`no` : array of numbers
`name` : array of names
`p` : array of processing times
`r` : array of release dates (SiPSi only)
`d` : array of due dates
`w` : array of tardiness weights

The length of the arrays `no`, `name`, `p`, (`r`,) `d` and `w` must be at least `n`. It assumes that the objective is total weighted tardiness, and the earliness weights are automatically set to 0. As SiPS_TWT_set_job(), missing jobs are automatically generated.

Any of the arrays except `no` can be `NULL`. If the specified jobs already exist, the data corresponding to the unspecified arguments are kept unchanged. Otherwise, the unspecified data of jobs are initialized by their default values.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.8 SiPS_TWET_set_job

Synopsis

For SiPS:

```
int SiPS_TWET_set_job(sips *prob, int no, char *name,
                      int p, int d, cost_t ew, cost_t tw);
```

For SiPSi:

```
int SiPS_TWET_set_job(sips *prob, int no, char *name,
                      int p, int r, int d, cost_t ew, cost_t tw);
```

Description

This function is same as SiPS_TWT_set_job() except that the objective is assumed to be total weighted earliness-tardiness and hence it specifies the earliness weight `ew` as well as the tardiness weight `tw`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.1.9 SiPS_TWET_set_jobs

Synopsis

For SiPS:

```
int SiPS_TWET_set_jobs(sips *prob, int n, int *no, char **name,
                       int *p, int *d, cost_t *ew, cost_t *tw);
```

For SiPSi:

```
int SiPS_TWET_set_jobs(sips *prob, int n, int *no, char **name,
                       int *p, int *r, int *d, cost_t *ew, cost_t *tw);
```

Description

This function is same as SiPS_TWT_set_jobs() except that the objective is assumed to be total weighted earliness-tardiness and hence it specifies the arrays of the earliness weights `ew` and the tardiness weights `tw`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.1.10 SiPS_delete_job

Synopsis

```
int SiPS_delete_job(sips *prob, int no);
```

Description

This function deletes a job whose number is `no` from `prob`. The remaining jobs are automatically renumbered from 0.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.11 SiPS_delete_jobs

Synopsis

```
int SiPS_delete_job(sips *prob, int n, int *no);
```

Description

This function deletes multiple jobs from `prob` at the same time. The array `no` specifies the job numbers to be deleted and `n` the array size. The remaining jobs are automatically renumbered from 0.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.12 SiPS_delete_all_jobs

Synopsis

```
int SiPS_delete_all_job(sips *prob);
```

Description

This function deletes all jobs from `prob`.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.13 SiPS_set_cost_function

Synopsis

```
int SiPS_set_cost_function(sips *prob, cost_t (*f)(int, int));
```

Description

This function specifies the user-defined cost function `f`. This cost function takes two arguments, the job index i and the time t, and returns $f_i(t)$. For example, the following `f()` has the same meaning as $(d_0, \alpha_0, \beta_0) = (10, 3, 1)$, $(d_1, \alpha_1, \beta_1) = (55, 2, 2)$, $(d_2, \alpha_2, \beta_2) = (40, 1, 3)$,

```
01: cost_t f(int i, int t)
02: {
03:  cost_t v;
04:  switch (i) {
05:  default:
06:  case 0:
07:    v = 3*max(10-t, 0) + 1*max(t-10, 0);
08:    break;
09:  case 1:
10:    v = 2*max(55-t, 0) + 2*max(t-55, 0);
11:    break;
12:  case 2:
13:    v = 1*max(40-t, 0) + 3*max(t-40, 0);
14:    break;
15:  }
16:  return(v);
17:}
```

If the user-defined cost function is specified, due dates, earliness weights and tardiness weights are ignored. (The due dates may be exploited in the tie-breaking rule.) The cost function can be deleted by specifying `NULL` as `f`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.1.14 SiPS_set_cost_function_long

Synopsis

```
int SiPS_set_cost_function_long(sips *prob,
                                cost_t (*f)(sips *, int, int));
```

Description

This function is the same as SiPS_set_cost_function() except that the cost function takes three arguments: the first argument is the pointer to the problem instance.

3.1.15 SiPS_set_horizon

(SiPSi only)

Synopsis

```
int SiPS_set_horizon(sips *prob, Tmax);
```

Description

By default, SiPS/SiPSi automatically calculates the scheduling horizon. However, SiPSi cannot do this when the user-defined cost function is specified by SiPS_set_cost_function(), In this case, the end of scheduling horizon `Tmax` must be specified by this function. `Tmax` is ignored when the user-defined cost function is not specified.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.16 SiPS_set_cost_type

(valid only when compiled with `COST_REAL`)

Synopsis

```
int SiPS_set_cost_type(sips *prob, int type);
```

"type" can be:
`SIPS_COST_INTEGER` : the objective function is integer-valued.
`SIPS_COST_REAL` : the objective function is real-valued.

Description

When compiled with `COST_REAL`, the objective function becomes real-valued. However, the objective function is assumed to be integer-valued even in this case if `SiPS_set_cost_type(prob, SIPS_COST_INTEGER)` is called.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.1.17 SiPS_set_initial_sequence

Synopsis

```
int SiPS_set_initial_sequence(sips *prob, int n, int *seq);
```

Description

This function specifies a job sequence to obtain an initial upper bound of the objective value. The heuristics in SiPS/SiPSi may not work efficiently when the job cost function is specified by SiPS_set_cost_function(). In this case, try this function. `seq` is the array of job numbers and `n` is the length of `seq`. If `n` is not equal to the number of jobs, SiPS/SiPSi automatically converts the sequence

into a feasible one. The initial sequence is deleted if the function is called with `n=0`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.2 Access the Instance

3.2.1 SiPS_get_problem_size

Synopsis

```
int SiPS_get_problem_size(sips *prob);
```

Description

This function simply returns the number of jobs in the problem instance.

Returns

number of jobs	: success
negative value	: fail

3.2.2 SiPS_get_job

Synopsis

For SiPS:

```
int SiPS_get_job(sips *prob, int no, int *p, int *d,
                 cost_t *ew, cost_t *tw);
```

For SiPSi:

```
int SiPS_get_job(sips *prob, int no, int *p, int *r, int *d,
                 cost_t *ew, cost_t *tw);
```

Description

This function retrieves the data of a single job specified by `no`.

`prob`	: problem object
`no`	: job number (starts from 0)
`p`	: processing time
`r`	: release date (SiPSi only)
`d`	: due date
`w`	: tardiness weight

Any of the arguments except `prob` and `no` can be `NULL`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.2.3 SiPS_get_jobs

Synopsis

For SiPS:

```
int SiPS_get_jobs(sips *prob, int n, int *no, int *p, int *d,
                  cost_t *ew, cost_t *tw);
```

For SiPSi:

```
int SiPS_get_jobs(sips *prob, int n, int *no, int *p, int *r, int *d,
                  cost_t *ew, cost_t *tw);
```

Description

This function retrieves the data of `n` jobs whose numbers are specified by `no`.

`prob`	: problem object
`n`	: number of jobs to be retrieved
`no`	: array of job numbers
`name`	: array of names
`p`	: array of processing times
`r`	: array of release dates (SiPSi only)
`d`	: array of due dates
`w`	: array of tardiness weights

The length of the arrays `no`, `name`, `p`, (`r`,) `d`, and `w` must be at least `n`.
Any of the arrays except `no` can be `NULL`.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.2.4 SiPS_print_jobs

Synopsis

```
int SiPS_print_jobs(sips *prob);
```

Description

This function displays the problem instance data via the standard output.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.3 Solver Parameters

3.3.1 SiPS_set_int_param

Synopsis

```
int SiPS_set_int_param(sips *prob, int type, int value);
```

Description

This function sets the parameter `type` to the integral value `value`. For the available parameters, please refer to Chapter 4.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.3.2 SiPS_set_real_param

Synopsis

```
int SiPS_set_real_param(sips *prob, int type, double value);
```

Description

This function sets the parameter `type` to the real value `value`. For the available parameters, please refer to Chapter 4.

Returns

`SIPS_OK`	: success
`SIPS_FAIL`	: fail

3.4 Solve the Instance

3.4.1 SiPS_solve

Synopsis

```
int SiPS_solve_problem(sips *prob);
```

Description

This function solves the problem instance. Whether it is optimally solved or not can be checked by its return value.

Returns

`SIPS_SOLVED`	: optimally solved
`SIPS_TIMELIMIT`	: reached time limit
`SIPS_MEMLIMIT`	: reached memory limit
`SIPS_FAIL`	: fail (invalid problem instance)

3.5 Access the Solution

3.5.1 SiPS_get_solution

Synopsis

```
int SiPS_get_solution(sips *prob, cost_t *f, int *no, int *c);
```

Description

This function returns the optimal (or best) solution obtained after the call of SiPS_solve().

`prob` : problem object
`f` : objective value
`no` : array of job numbers
`c` : array of completion times

The sequence `no[0]`, `no[1]`, ... gives the processing order of jobs in the solution, and `c[0]`, `c[1]`, ... are the completion times of the first job, the second job, and so on, respectively.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.5.2 SiPS_get_cputime

Synopsis

```
int SiPS_get_cputime(sips *prob, double *cputime);
```

Description

This function returns the CPU time in seconds spent for solving the problem instance.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

3.5.3 SiPS_print_solution

```
int SiPS_print_solution(sips *prob);
```

Description

This function displays the solution via the standard output.

Returns

`SIPS_OK` : success
`SIPS_FAIL` : fail

Chapter 4

Available Parameters

4.1 Common Parameters

4.1.1 SIPS_I_VERBOSE (integer)

Verbose level. The default value is 0.

0: silent
1: medium
2: verbose

4.1.2 SIPS_R_EPS (double)

The upper bound of the relative error. The default value is 10^{-5}.

4.1.3 SIPS_R_FUNCEPS (double)

In SiPSi, the job cost functions are stored as piecewise linear functions to compute upper bounds efficiently. This parameter specifies the upper bound of the relative error in piecewise linear functions. The default value is 10^{-8}.

4.1.4 SIPS_R_TIMELIMIT (double)

The time limit in seconds. A negative value means no time limit. The default value is -1 (no time limit).

4.1.5 SIPS_I_MAXMEMORY (integer)

The maximum memory space in MB for storing the network structure. The default value is 1536 (MB).

4.1.6 SIPS_I_WARMSTART (integer)

If this option is set to `SIPS_TRUE`, the optimal solution and Lagangian multipliers obtained at the previous call of SiPS_solve() are used as the initial upper bound and the initial multipliers, respectively. The default value is `SIPS_FALSE`.

4.1.7 SIPS_I_DUMMYJOBS (integer)

The maximum length of dummy jobs (SiPSi only). If it is set to 0, the maximum processing time of ordinary jobs is used. The default value is 0.

4.1.8 SIPS_I_SEARCHTYPE (integer)

The local search algorithm for the upper bound computation. `SIPS_LS_COMBINED_A` applies the enhanced dynasearch only at the end of Stage 2 and at the last iteration of Stage 3. `SIPS_LS_COMBINED_B` applies the enhanced dynasearch throughout Stage 3. The default value is `SIPS_LS_EDYNA` (SiPS) or `SIPS_LS_COMBINED_A` (SiPSi).

`SIPS_LS_NONE` (0)	: nothing
`SIPS_LS_DYNA` (1)	: dynasearch
`SIPS_LS_COMBINED_A` (2)	: dynasearch and enhanced dynasearch (type A)
`SIPS_LS_COMBINED_B` (3)	: dynasearch and enhanced dynasearch (type B)
`SIPS_LS_EDYNA` (4)	: enhanced dynasearch

4.1.9 SIPS_I_DPSIZE (integer)

The maximum number of absent jobs to switch between two heuristics based on dynamic programming and on greedy insertion, respectively. If the number of absent jobs exceeds this value, greedy insertion is adopted. The default value is 12 (SiPS) or 8 (SiPSi).

4.1.10 SIPS_I_TIEBREAK (integer)

The tie-breaking rule in the dominance check. The default value is `SIPS_TIEBREAK_EDD`.

`SIPS_TIEBREAK_HEURISTIC` (0)	: order in the best solution at the end of Stage 1
`SIPS_TIEBREAK_RHEURISTIC` (1)	: reverse order of the best solution
`SIPS_TIEBREAK_EDD` (2)	: EDD order
`SIPS_TIEBREAK_SPT` (3)	: SPT order
`SIPS_TIEBREAK_LDD` (4)	: LDD order
`SIPS_TIEBREAK_LPT` (5)	: LPT order
`SIPS_TIEBREAK_WSPT` (6)	: WSPT order
`SIPS_TIEBREAK_WLPT` (7)	: WLPT order
`SIPS_TIEBREAK_RELEASE` (8)	: nondecreasing order of release dates (SiPSi only)

4.2 Stages 1 and 2

4.2.1 SIPS_I_MINUPDATE (integer)

The lower bound must be updated by the conjugate subgradient algorithm at least (the number of jobs)/`SIPS_I_MINUPDATE` times. When the terminal condition specified by SIPS_I_TERMITER1 and SIPS_I_TERMRATIO1, or SIPS_I_TERMITER2 and SIPS_I_TERMRATIO2 are satisfied, the subgradient

algorithm is terminated if the lower bound is updated at least (the number of jobs)/`SIPS_I_MINUPDATE` times. The default value is 10.

4.3 Stage 1

4.3.1 SIPS_I_SKIPSTAGE1 (integer)

Skip Stage 1 if `SIPS_TRUE`. The default value is `SIPS_FALSE`.

4.3.2 SIPS_R_MULTIPLIER (double)

The initial Lagrangian multiplier. The default value is 0.0.

4.3.3 SIPS_R_INITSTEP1 (double)

The initial step size for the conjugate subgradient algorithm. The default value is 1.0.

4.3.4 SIPS_R_MAXSTEP1 (double)

The maximum step size for the conjugate subgradient algorithm. The default value is 1.0.

4.3.5 SIPS_I_SHRINKITER1 (integer)

The number of non-updating iterations to reduce the step size in the conjugate subgradient algorithm. The default value is 2.

4.3.6 SIPS_R_SHRINK1 (double)

The ratio of the reduction of the step size in the conjugate subgradient algorithm. The default value is 0.9 (SiPS) or 0.95 (SiPSi).

4.3.7 SIPS_R_EXPAND1 (double)

The ratio of the increase of the step size in the conjugate subgradient algorithm. The default value is 1.25 (SiPS) or 1.15 (SiPSi).

4.3.8 SIPS_R_INITTERMITER1 (double)

The terminal condition for the conjugate subgradient algorithm. The algorithm is terminated if the lower bound is never updated for more than (the number of jobs)/`SIPS\_I\_INITTERMITER1` iterations. In other words, the algorithm is not terminated for at least this number of iterations even if the lower bound is not updated at all. The default value is 3.0 (SiPS) or 0.75 (SiPSi).

4.3.9 SIPS_R_TERMITER1 (double)

The terminal condition for the conjugate subgradient algorithm. The default value is 4.0 (SiPS) or 1.0 (SiPSi).

4.3.10 SIPS_R_TERMRATIO1 (double)

The terminal condition for the conjugate subgradient algorithm. The default value is 0.02.

4.3.11 SIPS_I_UBUPDATE1 (integer)

Whether the upper bound is updated (`SIPS_TRUE`) or not (`SIPS_FALSE`) at the end of Stage 1. The default value is `SIPS_TRUE` (SiPS) or `SiPS_FALSE` (SiPSi).

4.4 Stage 2

4.4.1 SIPS_R_INITSTEP2 (double)

The initial step size for the conjugate subgradient algorithm. The default value is 1.0.

4.4.2 SIPS_R_MAXSTEP2 (double)

The maximum step size for the conjugate subgradient algorithm. The default value is 1.0.

4.4.3 SIPS_I_SHRINKITER2 (integer)

The number of non-updating iterations to reduce the step size in the conjugate subgradient algorithm. The default value is 2.

4.4.4 SIPS_R_SHRINK2 (double)

The ratio of the reduction of the step size in the conjugate subgradient algorithm. The default value is 0.9 (SiPS) or 0.95 (SiPSi).

4.4.5 SIPS_R_EXPAND2 (double)

The ratio of the increase of the step size in the conjugate subgradient algorithm. The default value is 1.3 (SiPS) or 1.2 (SiPSi).

4.4.6 SIPS_R_INITTERMITER2 (double)

The terminal condition for the conjugate subgradient algorithm. The algorithm is terminated if the lower bound is never updated for more than (the number of jobs)/`SIPS\_I\_INITTERMITER2` iterations. In other words, the algorithm is not terminated for at least this number of iterations even if the lower bound is not updated at all. The default value is 1.0 (SiPS) or 0.5 (SiPSi).

4.4.7 SIPS_R_TERMITER2 (double)

The terminal condition for the conjugate subgradient algorithm. The default value is 4.0 (SiPS) or 1.0 (SiPSi).

4.4.8 SIPS_R_TERMRATIO2 (double)

The terminal condition for the conjugate subgradient algorithm. The default value is 0.002.

4.4.9 SIPS_I_UBITERATION (integer)

The cycle (number of iterations) of the upper bound computation in the conjugate subgradient algorithm. The deafult value is 10 (SiPS) or 50 (SiPSi).

4.5 Stage 3

4.5.1 SIPS_I_SECSWITCH (integer)

The occupated memory size in MB to enter the sectioning mode in Stage 3. If it is negative, the algorithm never enters this mode. The default value is 16 (SiPS) or 6 (SiPSi).

4.5.2 SIPS_R_SECINIRATIO (double)

4.5.3 SIPS_R_SECRATIO (double)

In the sectioning mode, the tentative upper bound is changed, by default, as

$$(\text{lower bound}) + 2\Delta,\ (\text{lower bound}) + 3\Delta,\ (\text{upper bound}),$$

where Δ =((lower bound)-(upper bound))/4. If `SIPS_R_SECRATIO` is specified and larger than zero, the tentative upper bound is increased from

$$(\text{lower bound}) + \texttt{SIPS\_R\_SECINIRATIO} \cdot (\text{upper bound})$$

by

$$\texttt{SIPS\_R\_SECRATIO} \cdot (\text{upper bound}).$$

When `SIPS_R_SECINIRATIO` is not greater than zero, it is chosen as equal to `SIPS_R_SECRATIO`.

4.5.4 SIPS_I_BISECTION (integer)

Turn on (`SIPS_TRUE`) or off (`SIPS_FALSE`) the bisection mode. The default value is `SiPS_FALSE` (no bisection).

4.5.5 SIPS_I_MAXMODIFIER (integer)

The maximum number of modifiers to be added at one iteration. The default value is 3.

4.5.6 SIPS_I_MODSWITCH (integer)

The algorithm switches the job selection strategy from Type I to Type II for recovering constraints in Stage 3, if memory usage exceeds `SIPS_I_MODSWITCH` (MB). If it is negative, the algorithm only uses the Type I strategy. The default value is 0 (always Type II).

4.5.7 SIPS_I_ZEROMULTIPLIER (integer)

Zero multipliers are used together with those obtained in Stage 2 (`SIPS_TRUE`) or are not used (`SIPS_FALSE`). This option in general increases the CPU time, but may be effective especially for the total (weighted) tardiness problem with release dates. The default value is `SIPS_FALSE`.

4.5.8 SIPS_I_UBUPDATE3 (integer)

Whether the upper bound is updated (`SIPS_TRUE`) or not (`SIPS_FALSE`) at every iteration of Stage 3. The default value is `SIPS_FALSE`.

Chapter 5

Miscellany

5.1 Data File Format

Any characters after # are regarded as a comment. The first line specifies the number of jobs (n). The second and later lines specify job data. Each single line corresponds to a single job (jobs are numbered automatically from zero) and each column should be separated by white space(s). The line formats for SiPS and SiPSi differ from each other as follows.

5.1.1 SiPS

Each line is composed of two to four columns. Four columns specify the processing time, due date, earliness weight and tardiness weight in this order (total weighted earliness-tardiness objective). Three columns specify the processing time, due date and tardiness weight in this order, and the earliness weight is set to zero (total weighted tardiness objective). Two columns specify the processing time and due date. In this case the earliness weight is set to zero and the tardiness weight is set to one (total tardiness objective).

5.1.2 SiPSi

Each line is composed of three to five columns. Five columns specify the processing time, release date, due date, earliness weight and tardiness weight in this order (total weighted earliness-tardiness objective). Four columns specify the

```
# n
  3
# each line specifies a single job
# p  r  d ew tw
 10  0 10  3  1
 20  5 55  2  2
 30 10 40  1  3
```

Figure 5.1: Sample Data File for SiPSi

processing time, release date, due date and tardiness weight in this order, and the earliness weight is set to zero (total weighted tardiness objective). Three columns specify the processing time, release date and due date. In this case the earliness weight is set to zero and the tardiness weight is set to one (total tardiness objective).

5.2 Front-Ends sips/sipsi

The commands sips/sipsi are simple front-ends for SiPS/SiPSi that read instance data from a file and solve it.

5.2.1 Usage

To use sips/sipsi, type

```
sips [options] [file name]
```

or

```
sipsi [options] [file name]
```

If the file name is omitted, it reads the data from the standard input. Options should start from "-" and may take one argument. They are to specify parameters of SiPS/SiPSi.

5.2.2 Options

–h

Shows the usage.

–ti DOUBLE

Specifies the time limit in seconds (SIPS_R_TIMELIMIT).

–v INTEGER

Specifies the verbose level (SIPS_I_VERBOSE).

–lb

Computes a lower bound only.

–sk1

Specifies whether Stage 1 is skipped or not. (SIPS_I_SKIPSTAGE1).

–u DOUBLE

Specifies the initial Lagrangian multiplier (SIPS_R_MULTIPLIER).

–is1 DOUBLE

Specifies the initial step size for the conjugate subgradient algorithm in Stage 1 (SIPS_R_INITSTEP1).

–ms1 DOUBLE

Specifies the maximum step size for the conjugate subgradient algorithm in Stage 1 (SIPS_R_MAXSTEP1).

–si1 INTEGER

Specifies the number of non-updating iterations to reduce the step size in the conjugate subgradient algorithm in Stage 1 (SIPS_I_SHRINKITER1).

–s1 DOUBLE

Specifies the ratio of the reduction of the step size in the conjugate subgradient algorithm in Stage 1 (SIPS_R_SHRINK1).

–e1 DOUBLE

Specifies the ratio of the increase of the step size in the conjugate subgradient algorithm in Stage 1 (SIPS_I_EXPAND1)

–it1 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 1 (SIPS_R_INITTERMITER1).

–ti1 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 1 (SIPS_R_TERMITER1).

–r1 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 1 (SIPS_R_TERMRATIO1).

–is2 DOUBLE

Specifies the initial step size for the conjugate subgradient algorithm in Stage 2 (SIPS_R_INITSTEP2).

–ms2 DOUBLE

Specifies the maximum step size for the conjugate subgradient algorithm in Stage 2 (SIPS_R_MAXSTEP2).

–si2 INTEGER

Specifies the number of non-updating iterations to reduce the step size for the conjugate subgradient algorithm in Stage 2 (SIPS_I_SHRINKITER2).

–s2 DOUBLE

Specifies the ratio of the reduction of the step size in the conjugate subgradient algorithm in Stage 2 (SIPS_R_SHRINK2).

–e2 DOUBLE

Specifies the ratio of the increase of the step size in the conjugate subgradient algorithm in Stage 2 (SIPS_I_EXPAND2)

–it2 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 2 (SIPS_R_INITTERMITER2).

–ti2 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 2 (SIPS_R_TERMITER2).

–r2 DOUBLE

Specifies the terminal condition for the conjugate subgradient algorithm in Stage 2 (SIPS_R_TERMRATIO2).

–ub1/–nub1

Specifies whether the upper bound heuristics is applied or not at the end of Stage 1 (SIPS_I_UBUPDATE1).

–ubi INTEGER

Specifies the cycle (the number of iterations) of the upper bound computation in the conjugate subgradient algorithm in Stage 2 (SIPS_I_UBITERATION).

–nup INTEGER

Specifies the minimum number of updates of the lower bound in Stage 1 and Stage 2 (SIPS_I_MINUPDATE).

–ls INTEGER

Specifies the local search algorithm for the upper bound computation (SIPS_I_SEARCHTYPE).

0: none (SIPS_LS_NONE)
1: dynasearch (SIPS_LS_DYNA)
2: dynasearch and enhanced dynasearch (type A) (SIPS_LS_COMBINED_A)
3: dynasearch and enhanced dynasearch (type B) (SIPS_LS_COMBINED_B)
4: enhanced dynasearch (SIPS_LS_EDYNA)

–dmy INTEGER

Specifies the number of dummy jobs (SIPS_I_DUMMYJOBS) (SiPSi only).

–tb INTEGER

Specifies the tie-breaking rule in the dominance check (SIPS_I_TIEBREAK).

0: order in the best solution at the end of Stage 1 (SIPS_TIEBREAK_HEURISTIC)
1: reverse order of the best solution (SIPS_TIEBREAK_RHEURISTIC)
2: EDD order (SIPS_TIEBREAK_EDD)
3: SPT order (SIPS_TIEBREAK_SPT)
4: LDD order (SIPS_TIEBREAK_LDD)
5: LPT order (SIPS_TIEBREAK_LPT)
6: WSPT order (SIPS_TIEBREAK_WSPT)
7: WLPT order (SIPS_TIEBREAK_WLPT)
8: nondecreasing order of release dates (SIPS_TIEBREAK_RELEASE) (SiPSi only)

–mem INTEGER

Specifies the maximum memory space (in MB) for storing the network structure (SIPS_I_MAXMEMORY).

NOTE: This option considerably affects the algorithm. The default value is 1536 (MB).

–ssw INTEGER

Specifies the memory size to switch to the sectioning mode in Stage 3 (SIPS_I_SECSWITCH).

–sir DOUBLE

Specifies the initial ratio to increase the tentative upper bound in Stage 3 (SIPS_R_SECINIRATIO).

–sr DOUBLE

Specifies the ratio to increase the tentative upper bound in Stage 3 (SIPS_R_SECRATIO).

–bi/–nbi

Turns on/off the bisection mode in Stage 3 (SIPS_I_BISECTION).

–mod INTEGER

Specifies the maximum number of modifiers to be added at one iteration in Stage 3 (SIPS_I_MAXMODIFIER).

–msw INTEGER

Specifies the memory size to switch the job selection strategy for adding modifiers in Stage 3 (SIPS_I_MODSWITCH).

–zmp/–nzmp

Turn on/off zero multipliers in Stage 3 (SIPS_I_ZEROMULTIPLIER).

–ub3/–nub3

Specifies whether the upper bound is updated or not at every iteration of Stage 3 (SIPS_I_UBUPDATE3).