# BAB 2
# LANDASAN TEORI

## 2.1 Constraint Satisfaction Problems (CSP)

CSP merupakan sebuah tipe permasalahan yang masuk kedalam bidang *artificial intelligence*, dimana tujuan penyelesaiannya adalah mendapatkan nilai *assignment* untuk setiap variabel berdasarkan *constraint* yang ditetapkan sehingga mendapatkan solusi yang layak [23]. Permasalahan-permasalahan yang dikategorikan seperti CSP antara lain seperti *Job-shop Scheduling*, *timetabling*, dan permainan sudoku. Terdapat beberapa komponen utama dalam penyelesaian CSP sebagai berikut.

1. Variables : Merepresentasikan sebuah entitas yang perlu di-*assign* sebuah nilai dalam bentuk apapun. Setiap variabel memiliki domain yang bersangkutan, yang berisi kumpulan dari nilai yang memiliki kemungkinan untuk dimasukkan kedalam sebuah variabel.

2. Domains : Merupakan kumpulan atas nilai yang dapat diambil oleh setiap variabel. Sebuah domain dapat berbentuk diskrit yang memiliki jumlah terbatas yang dapat dihitung, juga dapat berbentuk kontinu dimana data tidak terbatas dan cenderung tidak tetap.

3. Constraint : Merupakan sebuah kendala yang membatasi kombinasi antar nilai untuk di-*assign* kedalam setiap variabel. Sebuah *constraint* dapat berupa *unary* (hanya berhubungan dengan 1 variabel), *binary* (dapat menghubungkan 2 variabel), dan bisa juga mengaitkan lebih dari 2 variabel

Dalam permasalahan CSP, setiap variabel memiliki *domain* atas seluruh kemungkinan nilai untuk diambil dan terdapat *constraint* yang harus dipatuhi oleh setiap variabel untuk pengambilan nilai dalam rangka mencari solusi yang valid [23]. Berbeda dengan istiliah *optimization*, dimana CSP tidak memiliki sebuah *objective function* yang digunakan untuk menilai setiap *possible solution* berdasarkan bobot bobot yang ditetapkan [24], jadi CSP hanya mencari solusi yang layak dengan meminimalisir jumlah pelanggaran terhadap *constraint* yang ditetapkan.

Dalam penyelesaian CSP ataupun *optimization*, umumnya terdapat dua jenis *constraint*, yang terbagi menjadi *soft constraint* dan *hard constraint*. Sebuah

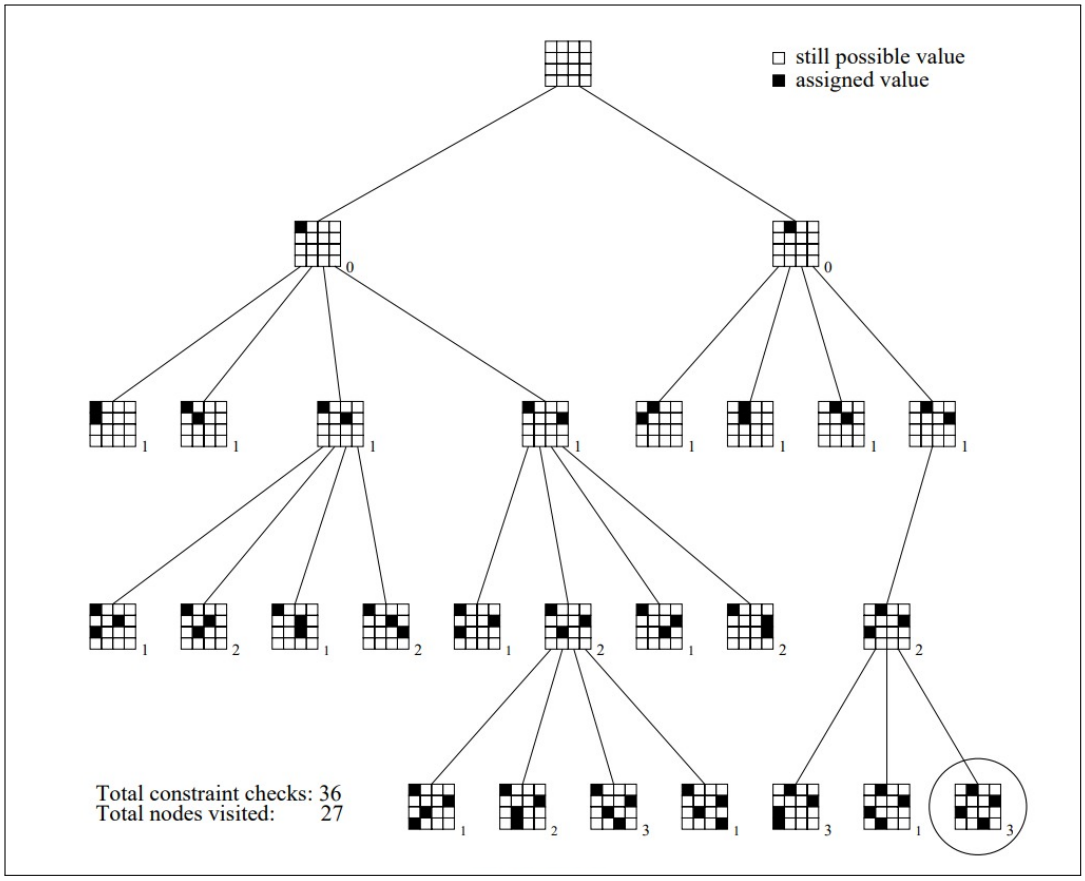
batasan dikategorikan sebagai *hard constraint* ketika batasan tersebut harus dipatuhi tanpa ada pelanggaran pada setiap solusi yang dihasilkan [25]. Sementara itu pelanggaran terhadap *soft constraint* masih dapat ditoleransi, namun pelanggaran terhadap *soft constraint* umumnya akan memberikan *penalty* terhadap *objective function*, sehingga semakin banyak pelanggaran terhadap *soft constraint* maka semakin jauh solusi tersebut dari *optimal solution* [25]. Dengan demikian meskipun permasalahan dengan tipe CSP tidak memiliki *objective function* secara eksplisit, ketika permasalahan yang ingin diselesaikan mengandung batasan berupa *soft constraint*, maka mekanisme yang serupa dengan *objective function* akan diterapkan untuk mencari solusi dengan meminimalisir pelanggaran terhadap *soft constraint*.

Dalam menyelesaikan CSP, terdapat dua teknik umum yang digunakan, yaitu *Tree search* dan *Constraint Propagation* [26].

### 2.1.1 Tree Search

Dalam pendekatan ini, penliaian pada setiap variabel dilakukan satu per satu dalam bentuk sebuah *tree*. Sebuah *tree* akan memiliki node berupa setiap variabel yang nilainya akan diassign dengan nilai yang tersedia pada domain. *Parent node* dari *tree* tersebut akan merepresentasikan seluruh variabel yang belum mendapatkan nilai apapun. Lalu pendekatan ini akan melakukan *branching* untuk mengeksplorasi seluruh kemungkinan *assignment* pada setiap variabel sambil memastikan tidak ada *constraint* yang dilanggar. Ketika node yang sedang dieksplor sudah tidak ada nilai yang dapat di-*assign*, maka node tersebut adalah yang terakhir pada setiap branch, dengan demikian proses selanjutnya adalah mengeksplor branch lainnya menggunakan teknik *depth-first search* hingga setiap variabel berhasil mendapatkan *value* tanpa melanggar *constraint* apapun.

Berikut akan dijelaskan contoh penerapan *Tree Search* dalam menyelesaikan permasalahan *N-Queen Problem*. Permasalahan N-Queen dapat diibaratkan dalam sebuah papan catur dengan jumlah kotak N x N, dan terdapat N jumlah Queen (bidak catur) yang perlu ditempatkan dengan aturan setiap baris hanya dapat diisi oleh seorang *queen*, dan setiap queen tidak boleh berada digaris yang sama baik secara vertikal, horizontal, ataupun diagonal.




Gambar 2.1. Tree Search on N-Queen Problem

Sumber: [26]

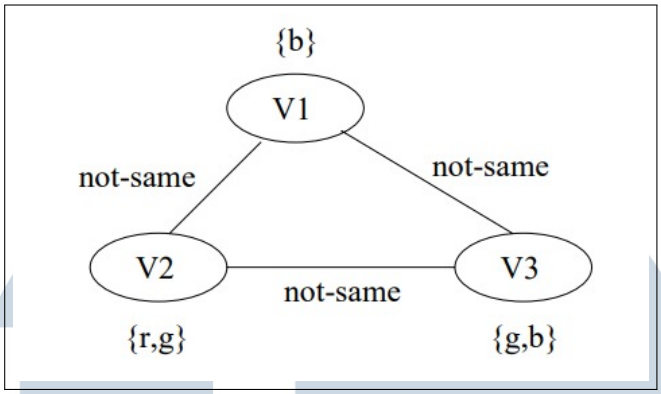
Terlihat pada gambar diatas, pada parent node berisi kumpulan dari variabel (kotak pada papan catur) belum mendapatkan nilai (boolean keberadaan queen), lalu akan dimulai tahap branching dengan meng-*assign* setiap *possible value* untuk variabel secara berurutan. Jika *node* yang terbentuk melanggar salah satu constraint, maka proses *branching* tidak akan dilanjutkan pada node tersebut. Seperti gambar diatas, pada node level 1 urutan pertama dan kedua tidak dilanjutkan karena node tersebut telah melanggar *constraint* dimana terdapat *queen* yang diletakkan berdampingan secara diagonal. Proses seperti itu diteruskan hingga mendapatkan node dimana setiap variabel berhasil mendapatkan *value* tanpa melanggar salah satu *constraint*, atau ketika sudah selesai melakukan eksplorasi tanpa mendapatkan *feasible solution*.

### 2.1.2 Constraint Propagation

Teknik ini bertujuan untuk mengubah permasalahan CSP menjadi permasalahan yang masih serupa, namun lebih mudah untuk diselesaikan. Proses ini dilakukan dengan cara memperkecil nilai pada domain yang mungkin untuk di-*assign* dengan mempertimbangkan setiap *constraint*. Jadi ketika sebuah variabel telah mendapatkan nilai, maka proses ini akan mengidentifikasi keberadaan *inconsistent values*, singkatnya *inconsistent values* merupakan nilai-nilai yang sudah tidak possible untuk diambil bersamaan dengan nilai yang sementara sudah di-*assign* kedalam variabel tersebut.

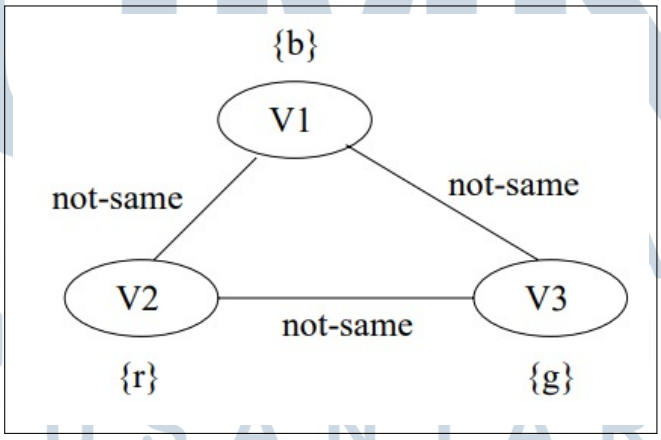
Dalam sebuah CSP telah disebutkan bahwa umumnya terdapat dua jenis *constraint*, yaitu *unary* dan *binary*. Sebauh permasalahan CSP dinyatakan *node consistent* ketika setiap *value* pada domain tidak melanggar constraint dari sebuah variabel. Jika terdapat value yang jika di-*assign* terhadap sebuah variabel tidak memenuhi *unary constraint* pada variabel tersebut, maka value itu akan dianggap *redundant* dan dapat dikeluarkan dari proses eksplorasi solusi. Lalu terdapat derajat konsistensi yang lebih tinggi, yang disebut dengan *arc consistent* dimana terdapat dua variabel a dan b, ketika setiap value yang diambil pada variabel a memiliki *consistent value* untuk diambil pada variabel b, maka hubungan/*arc* antar variabel a dan b dianggap *arc consistent*

Dalam konteks *arc consistent*, dapat dibayangkan dalam permasalahan *map-colouring problem*, dimana terdapat sebuah graph dengan vertex yang merepresentasikan setiap variabel dengan masing masing domain yang berisi kemungkinan warna yang diambil (sudah node-consistent). Edge dari graf tersebut merepresentasikan *binary constraint* dimana setiap graph yang berdampingan tidak boleh memiliki warna yang sama.




Gambar 2.2. Constraint Propagation on Map Colouring Problem (Not Consistent)
Sumber: [26]

Pada graph diatas terdapat beberapa arc yang terbentuk, dan terdapat sebuah arc yang masih belum *arc-consistent*. Pada G(V3,V1) dimana ketika V3 mendapatkan nilai b dan pada V1 hanya memiliki nilai b, maka *arc* tersebut dinyatakan tidak konsisten. Meskipun begitu, bukan berarti G(V1,V3) juga tidak konsisten, karena ketika V1 mendapatkan nilai b, terdapat *consistent value* g pada V3. Maka dari itu akan terdapat penghapusan value b pada domain V3. Akan tetapi setelah penghapusan value b pada domain V3, menyebabkan keberadaan *arc* yang tidak konsisten pada G(V2,V3) karena jika V2 mengambil nilai g, tidak ada konsisten value yang tersedia pada V3, oleh karena itu value g akan dihapus pada domain V2. Dengan demikian graf tersebut akan menjadi *arc consistent* seperti gambar berikut.




Gambar 2.3. Constraint Propagation on Map Colouring Problem (Consistent)
Sumber: [26]

## 2.2 Student Scheduling Problem (SSP)

SSP merupakan salah satu variasi dari permasalahan CSP dimana setiap kelas yang dibuka pada sebuah universitas dianggap menjadi sebuah *set of offering* dan merupakan sebuah domain dalam konteks CSP. Lalu mahasiswa sebagai entitas atau variabel akan mengambil kombinasi value dalam domain tersebut sehingga mendapatkan kelas yang harus dihadiri setiap minggu-nya [23].

*Input* yang digunakan dalam permasalahan SSP ini umumnya berupa sebuah group dari mahasiswa yang memiliki *requirement* yang sama, dimana *requirement* tersebut adalah mata kuliah yang perlu/ingin diambil oleh setiap mahasiswa mahasiswa. Lalu terdapat *set of offering* yang diterbitkan oleh pihak universitas, serta berbagai constraint yang ditetapkan saat melakukan pemodelan. Sementara itu *output* yang diharapkan berupa kombinasi value yang berasal dari *set of offering* untuk setiap mahasiswa yang memenuhi *requirement* serta tidak melanggar *constraint* yang ditetapkan [23].

Dalam permasalahan SSP, terdapat beberapa constraint yang telah dipertimbangkan dari beberapa penelitian dengan permasalahan yang serupa, antara lain sebagai berikut [15].

1. Setiap mahasiswa hanya dapat menjalankan satu kelas pada setiap rentang waktu (C1)
2. Preferensi kelas pilihan mahasiswa harus terpenuhi (C2)
3. Terdapat kapasitas kelas maksimal yang tidak boleh dilebihi (C3)
4. Terdapat jumlah minimum mahasiswa pada setiap kelas untuk dipenuhi (C4)
5. Terdapat batas maksimal bobot kelas untuk diambil dalam satu semester pada setiap mahasiswa (C5)
6. Terdapat kelas yang wajib diambil oleh mahasiswa (C6)
7. Mahasiswa memiliki preferensi waktu pelaksanaan kelas (C7)
8. Jumlah mahasiswa pada setiap kelas dalam sebuah mata kuliah perlu dijaga keseimbangan-nya (C8)
9. Meminimalisir perpindahan antar gedung (C9)
10. Terdapat prioritas pada mata kuliah tertentu (C10)

11. Memperhatikan keseimbangan bobot matakuliah pada setiap hari-nya (C11)
12. Terdapat jeda waktu antar kelas untuk mahasiswa beristirahat (C12)
13. Mahasiswa memiliki jam sibuk untuk tidak mengikuti kelas pada waktu tertentu (C13)
14. Pengelompokkan mahasiswa dimana setiap anggota kelompok mendapatkan kelas yang sama (C14)

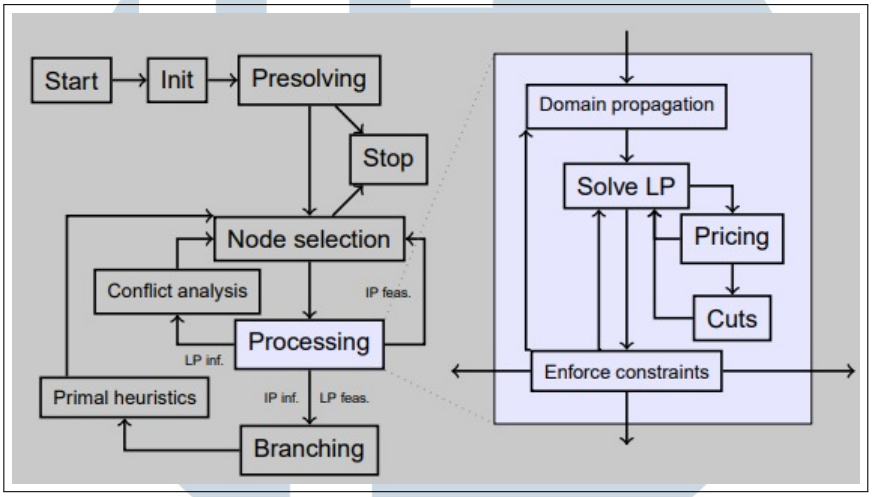
Dari semua *constraint* yang disebutkan, aturan yang harus dipatuhi seperti *time-conflict* dan kapasitas kelas akan dikategorikan sebagai *hard constraint*, sedangkan aturan yang dapat ditoleransi seperti waktu pelaksanaan kelas akan dianggap sebagai *soft constraint*. Penerapan *constraint* bisa berbeda pada setiap universitas mengikuti kebijakan yang diterapkan.

## 2.3 Constraint Integer Programming (CIP)

CIP merupakan sebuah pendekatan gabungan antara Mixed Integer Programming (MIP), Constraint Programming (CP), dan Boolean Satisfiability (SAT) untuk menyelesaikan permasalahan kompleks. Dengan penggabungan ketiga teknik, CIP memungkinkan constraint untuk direpresentasikan dalam beberapa bentuk seperti linear, integer, logical, hingga constraint yang lebih general seperti yang ditawarkan dalam pendekatan CP. Dalam pencarian solusi, CIP mengkombinasikan teknik yang diterapkan pada masing masing pendekatan, seperti *cutting plains* pada MIP, *domain propagation* pada CP, dan *conflict analysis* pada pendekatan SAT [27].

Dalam menerapkan pendekatan CIP, terdapat sebuah framework bernama SCIP (*Solving Constraint Integer Programs*) yang didasari dengan prosedur *branch-and-bound* [19]. Prosedur tersebut seperti namanya terbagi menjadi dua hal penting, yang pertama adalah *branching* dimana dalam konteks CSP pada awalnya kita memiliki sejumlah variabel yang harus diberi nilai dengan memperhatikan batasan yang ditentukan, konsep *branching* akan digunakan untuk membagi problem secara keseluruhan menjadi *subproblem* untuk mempersempit ruang solusi dalam bentuk sebuah *tree search*. Setelah itu, proses *bounding* mulai dilakukan dengan memberi batasan atas dan batasan bawah pada setiap node. Setelah mendapatkan batasan bawah dan batasan atas, maka akan terdapat proses *prunning* untuk menentukan subdomain mana yang perlu dieksplorasi lebih lanjut dengan membandingkan

batasan atas dan bawah pada setiap subdomain [28]. Dalam mengimplementasikan pendekatan *branch-and-bound*, SCIP menggunakan gabungan beberapa teknik yang terintegrasi dalam sebuah proses iterasi sebagai teknik pencarian solusi. Secara umum berikut merupakan alur framework SCIP dalam mendapatkan *optimum solution*.




Gambar 2.4. Flowchart pencarian solusi pada SCIP
Sumber: [27]

Proses dimulai dengan ***problem specification*** dimana *decision variable* dan *constraint* direpresentasikan, lalu kedua hal tersebut akan dimasukkan kedalam tahap ***presolving*** yang bermaksud untuk mentransofrmasikan permasalahan yang telah dispesifikasikan menjadi permasalahan yang lebih mudah untuk diselesaikan. Presolving dilakukan dengan beberapa hal seperti mengidentifikasi keberadaan constraint dan variabel yang redundant untuk dihapuskan, *tightening variable bounds* juga dilakukan ketika terdapat *constraint* yang memberi batasan nilai kepada sebuah variabel, dan beberapa hal lainnya terkait dengan variabel dan *constraint*. Lalu tahap pencarian solusi dimulai dengan membentuk *search tree* dan memilih *root node* untuk diproses pada iterasi pertama. Pada setiap iterasi, ***primal heuristic*** dipanggil pada setiap iterasi dalam rangka mengeksplor ruang solusi pada subproblem (state proses pencarian solusi) dengan cepat dan juga memiliki potensi untuk mendapatkan *feasible solution*. Selain itu dalam proses eksplorasi ruang solusi, *primal heuristic* umumnya dapat memperbaiki kualitas solusi sementara dari subproblem dengan cara seperti memodifikasi nilai yang telah diassign agar lebih optimal berdasarkan sebuah *objective function* sebelum memasuki proses ***node selection*** sebagai iterasi selanjutnya. Ketika memilih node untuk dieksplor,

proses ***domain propagation*** juga akan dilakukan untuk memperkecil domain untuk di-*assign* kedalam variabel berdasarkan *constraint* dan variabel sebelumnya yang telah mendapatkan *value* [19].

Setelah itu proses ***LP Solving*** akan dijalankan dengan tujuan mendapatkan LP Relaxation dari permasalahan saat ini. LP Relaxation didapatkan dengan menghilangkan batasan integralitas variabel, yang berguna untuk mendapatkan lower-bound dalam konteks branch-and-bound. Lower-bound didapatkan dengan melakukan relaksasi linear terhadap *constraint integer* dalam sebuah permasalahan yang memungkinkan sebuah variabel mendapatkan nilai fraksional, lalu dari permasalahan yang telah direlaksasi akan diambil nilai objektifnya sebagai *lower-bound*. Dalam tahap ini juga terdapat proses ***variable pricing*** yang akan dipanggil untuk memperbaiki nilai batas bawah dari sebuah subproblem dengan menambah variabel yang belum dipertimbangkan sebelumnya. Lalu terdapat proses ***cut separation*** dengan maksud untuk memperbaiki solusi yang dihasilkan melalui LP Relaxation dengan mengeksploitasi batasan integralitas sebuah variabel sehingga relaksasi linear dapat diperketat dan menghasilkan *lower-bound* yang lebih baik. Sedangkan nilai *upper-bound* akan diambil dari nilai *objective function* dari permasalahan original yang berbentuk integer. Kedua batas tersebut nantinya akan digunakan sebagai tahap *pruning* agar meminimalisir node yang perlu dieksplorasi [19].

Selama proses LP Relaxation dilakukan, terdapat beberapa *constraint* yang dilemaskan atau bahkan dibuang, maka dari itu ***constraint enforcement*** dilakukan untuk memastikan solusi yang ditemukan pada masa relaksasi tidak melanggar *hard-constraint*. Hasil relaksasi yang didapat bisa dilanjutkan dengan mengeksplorasi node lainnya atau membuat *branch* baru dari *node* yang sedang dieksplor saat ini. Jika hasil relaksasi dari node yang saat ini dieksplorasi menghasilkan *infeasibility*, maka node tersebut dapat dipangkas dan tidak perlu dilanjutkan, namun jika hasilnya *feasible* maka bisa mengeksplor node lainnya atau membuat *branch* baru untuk eksplorasi. Akan tetapi jika solusi yang dihasilkan *infeasible* dan tidak dapat melakukan *branching*, maka sebelum memilih node lain untuk dieksplorasi, akan dilakukan ***conflict analysis*** terlebih dahulu. *Conflict analysis* dilakukan dengan menganalisa penyebab solusi yang ditemukan *infeasiible* untuk menghindari konflik yang serupa pada node lainnya, dengan demikian dapat dilakukan *pruning* terhadap node-node yang berpotensi menimbulkan konflik [19].

Framework SCIP pada dasarnya sudah memiliki dokumentasi yang jelas yang dapat diakses melalui link https://www.scipopt.org/ mencakup juga prosedur

untuk instalasi dimana SCIP ini dapat digunakan sebagai *black-box solver* yang berarti pengguna hanya bisa mengetahui *input* dan *output*-nya saja, tanpa mengetahui proses-proses yang ada didalamnya. Selain itu tersedia juga beberapa interface yang dapat digunakan untuk menerapkan framework SCIP dalam bentuk *package* atau *library* yang dapat diakses oleh beberapa bahasa pemrograman, berikut merupakan tabel yang mengandung detail informasi dari masing masing interface yang tersedia.

Tabel 2.1. Daftar Interface untuk Framework SCIP

| **Interface** | **Link** | **Cust. Plugin** | **Build&Solve** | **Query Sol** | **Set Params** |
|---|---|---|---|---|---|
| C/C++-API | here | all | yes | yes | yes |
| Python/PySCIPOpt | here | all | yes | yes | yes |
| Julia/SCIP.jl | here | all | yes | yes | yes |
| Matlab | here | no | yes | no | yes |
| Java/JSCIPOpt | here | no | yes | yes | yes |
| AMPL | here | no | yes | no | yes |
| GAMS | here | no | yes | yes | yes |
| Rust | here | yes | yes | yes | yes |

Terlihat pada tabel 2.1 terdapat banyak interface yang dapat digunakan untuk menerapkan *framework* SCIP. Masing masing ada yang dapat diimplementasikan sebagai *package* atau *library* dalam berbagai bahasa pemorgraman seperti C++, Java, dan Python. Selain itu terdapat juga interface yang bisa menghubungkan framework *SCIP* ini dalam software atau sistem lainnya yang khusus bekerja sebagai pembuatan model *optimization* seperti AMPL, GAMS, Matlab, dan juga Rust. Namun terlihat juga pada keempat kolom terakhir, dimana tidak semua interface telah mendukung penggunaan fitur fitur dari *framework SCIP* secara keseluruhan mulai dari penggunaan *plugin*, perancangan dan eksekusi model penjadwalan, penarikan solusi, hingga beberapa parameter yang dapat disesuaikan secara manual.